08-18-02

Please type a plus sign (+) inside this box → [+]

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No.	3246.1
First Inventor or Application Identifier	Hubbell
Title	Methods, Systems and Computer Software for Designing and Synthesizing Sequence Arrays
Express Mail Label No.	EL474110579US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

1. [X] * Fee Transmittal Form *(e.g., PTO/SB/17)* *(Submit an original and a duplicate for fee processing)*
2. [X] Specification [Total Pages 32] *(preferred arrangement set forth below)*
   - Descriptive title of the Invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings *(if filed)*
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) *(35 U.S.C. 113)* [Total Sheets 4]
4. Oath or Declaration [Total Pages 3]
   - a. [X] Newly executed (original or copy)
   - b [ ] Copy from a prior application (37 C.F.R § 1.63(d)) *(for continuation/divisional with Box 16 completed)*
     - i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

5. [ ] Microfiche Computer Program *(Appendix)*
6. Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all necessary)*
   - a. [ ] Computer Readable Copy
   - b. [ ] Paper Copy (identical to computer copy)
   - c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. [X] Assignment Papers (cover sheet & document(s))
8. [ ] 37 C.F.R §3.73(b) Statement *(when there is an assignee)* [ ] Power of Attorney
9. [ ] English Translation Document *(if applicable)*
10. [X] Information Disclosure Statement (IDS)/PTO-1449 [X] Copies of IDS Citations
11. [ ] Preliminary Amendment
12. [X] Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
13. [ ] * Small Entity Statement(s) *(PTO/SB/09-12)* [ ] Statement filed in prior application, Status still proper and desired
14. [ ] Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
15. [X] Other: Appendix A; Appendix B

16. **If a CONTINUING APPLICATION,** *check appropriate box, and supply the requisite information below and in a preliminary amendment:*

[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No: _____/_____

Prior application information: Examiner _____ *Group / Art Unit:* _____

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

[X] *Customer Number or Bar Code Label* *(Insert Customer No. or Attach bar code label here)* 22886 or [ ] *Correspondence address below*

PATENT TRADEMARK OFFICE

Name					
Address					
City		State		Zip Code	
Country		Telephone		Fax	

Name (Print/Type)	Wei Zhou	Registration No. (Attorney/Agent)	44,419
Signature		Date	8/16/00

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

Express Mail Label No. EL474110579US

Attorney Docket Number: 3246.1

PATENT APPLICATION

METHODS, SYSTEMS AND COMPUTER SOFTWARE FOR DESIGNING AND SYNTHESIZING SEQUENCE ARRAYS

Inventor: **EARL A. HUBBELL**
a Citizen of the United States of America,
residing at 416 S. Genesee, Los Angeles,
California, United States of America

Assignee: **AFFYMETRIX, INC.**
3380 Central Expressway
Santa Clara, California 95051
a Delaware corporation

Status: Large Entity

Legal Department
Affymetrix, Inc.
3380 Central Expressway
Santa Clara, CA 95051

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the priority of U. S. Provisional Applications, Serial No. 60/149,510, filed on August 17, 1999, titled "Edge Minimization" and Serial No. 60/182,288, filed on February 14, 2000, titled " Lithographic Mask Design and Synthesis of Diverse Probes on a Substrate." The 60/149,510 and 60/182,288 applications are incorporated in their entity herein by reference for all purposes.

COPYRIGHT NOTICE

APPENDIX

Appendices A and B are included herewith and form a part of the disclosure.

BACKGROUND OF THE INVENTION

U.S. Patent No. 5,424,186 describes a pioneering technique for, among other things, forming and using high density arrays of molecules such as oligonucleotide, RNA, peptides, polysaccharides, and other materials. This patent is hereby incorporated by reference for all purposes. Arrays of oligonucleotides or peptides, for

example, are formed on the surface by sequentially removing a photoremovable group from a surface, coupling a monomer to the exposed region of the surface, and repeating the process. These techniques have been used to form extremely dense arrays of oligonucleotides, peptides, and other materials. Such arrays are useful in, for example, drug development, gene expression monitoring, genotyping, and a variety of other applications. The synthesis technology associated with this invention has come to be known as "VLSIPS^{TM} " or "Very Large Scale Immobilized Polymer Synthesis" technology. Despite the great success of the technique disclosed in the U.S. Patent No. 5,434,186, there is still a need for improved methods for large scale synthesis of polymers.

SUMMARY OF THE INVENTION

According to some aspects of the invention, methods, systems, and computer software are provided for improving the arrangement of specified features within complex patterns. One aspect of the invention concerns arranging the specified features to have a reduced number of differences between adjacent features (edges). The methods, systems, and computer software products are particularly suitable for designing and forming sequence arrays such as nucleic acid or peptide arrays.

In one aspect of the invention, computer implemented methods for arranging polymers for combinatorial synthesis of said polymers on a substrate are provided. In some embodiments, computer-implemented optimization steps for performing a travelling salesman optimization are performed to arrange polymers in an order such

that when such polymers are assigned spatial locations for synthesis, edge counts between synthesis sites are reduced to reduce errors during photodirected synthesis, such as diffraction, internal reflection, and scattering. As used herein, the term edge-count may be a weighted edge-count taking into account distances to cells leaking radiation.

In one particularly preferred embodiment of the invention, this travelling salesman optimization is carried out using a locally greedy insertion algorithm, although many other methods for performing a travelling salesman optimization are also suitable for at least some embodiments of the invention.

In another aspect of the invention, computer implemented methods for transforming a pre-existing assignment of polymers to spatial locations for synthesis into an assignment of polymers to spatial locations with reduced edge counts. In a preferred embodiment, such methods use a locally greedy algorithm to choose new spatial locations for the polymers. In a preferred embodiment, a locally greedy optimization is performed on either polymers or blocks of polymers. In some embodiments, the locally greedy optimization involves dividing polymers into a plurality of blocks, wherein each of the blocks contains one or more related polymers, and each of the blocks is to be assigned to one corresponding slot on the substrate, where a slot is a plurality of locations sufficient to contain the polymers in a block. The process may be repeated until all blocks are assigned. In a preferred embodiment, the blocks are first ordered randomly, to avoid poor initial arrangements of polymers. In the preferred embodiment, a subset of the blocks from the set of currently

unassigned blocks is selected, usually starting from the first unassigned block. The number of blocks in the subset may be adjusted by the user. Preferred ranges may include, 5-20, 20-100,100-500, 500-1000, 1000-10000, 10000-100000 blocks in a subset. Such ranges may be chosen by the user to adjust, for example, the running time of the methods. One block of the subset is assigned to an empty slot if this block is the block whose assignment to the empty slot results in the least edge count of all blocks possibly assigned to the slot.

This method is particularly useful for arranging oligonucleotide probes in a nucleic acid array that is manufactured using photodirected combinatorial synthesis using a set of masks or computer controlled micromirrors.

In another aspect of the invention, computer software products for arranging polymers for combinatorial synthesis of polymers on a substrate are provided. The computer software product contains: 1) computer program code for performing a travelling salesman optimization to arrange polymers in an order such that when such polymers are assigned spatial locations for synthesis, edge counts between synthesis sites are reduced; and 2) a computer readable medium for storing the codes.

In another aspect of the invention, computer software products for transforming a pre-existing assignment of polymers to spatial locations for synthesis into an assignment of polymers to spatial locations with reduced edge counts are provided. The computer software product contains computer program code for performing a locally greedy algorithm for assigning polymers to spatial locations, and a computer readable medium for storing the codes. In a preferred embodiment, the

computer software product contains program code for performing locally greedy optimization including computer program code for dividing polymers into a plurality of blocks, computer program code for unassigning such blocks from their current spatial locations, computer program code for selecting a subset of the blocks from unassigned blocks, and computer program code for assigning one block of the set to an empty slot if the block results in a least edge count among the blocks of the subset.

The computer software product may also contain program code for repeating the steps of selecting and assigning until all blocks are assigned. In some preferred embodiments, the computer software product may contain computer program code for randomly ordering unassigned blocks, and may contain computer software code for accepting a number of blocks in a subset.

Furthermore, a computer implemented method for robust arrangement problem (RAP) is also provided. Oligonucleotide arrays for monitoring gene expression may have certain number of probe pairs or probes devoted to any given gene. Local problems (flecks of dust, bubbles, defects) may occur on the array, and if the probes (pairs) are arranged adjacent to each other (these probes may be referred hereafter as non-robust, bad or adjacent), there may be no informative probes remaining for that gene if a defect occurs. The RAP is a probe distribution problem of arranging all the probes (pairs) on the chip, so that of the N (typically, 10, 15 or 20 pairs) probes (pairs) associated with any given gene, no more than K, such as 2, 3, 4 or 5, of them are within a radius R of each other.

In some embodiments, all non-robust probe pairs are removed from the chip as

blocks, leaving empty slots behind, and an equal number of robust probe pairs are chosen randomly and also removed, and then these blocks are replaced (almost) randomly into the slots, the number of new non-robust blocks will be reduced greatly (typically again cut to 1% of the former value). Computer software products containing code for performing the RAP steps are also provided. In preferred embodiments, a polymer (probe) arrangement software product performs the edge minimization and solves RAP.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

Figure 1 illustrates an example of a computer system that may be utilized to execute the software of an embodiment of the invention.

Figure 2 illustrates a system block diagram of the computer system of Fig. 1.

Figure 3 shows a process for a locally greedy optimization.

Figure 4 shows a process for using one embodiment of the software product of the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to

these embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention.

As will be appreciated by one of skill in the art, the present invention may be embodied as a method, data processing system or program products. Accordingly, the present invention may take the form of data analysis systems, methods, analysis software and etc. Software written according to the present invention is to be stored in some form of computer readable medium, such as memory, hard-drive, DVD ROM or CD ROM, or transmitted over a network, and executed by a processor.

Fig. 1 illustrates an example of a computer system that may be used to execute the software of an embodiment of the invention. Fig. 1 shows a computer system 1 that includes a display 3, screen 5, cabinet 7, keyboard 9, and mouse 11. Mouse 11 may have one or more buttons for interacting with a graphic user interface. Cabinet 7 preferably houses a CD-ROM or DVD-ROM drive 13, system memory and a hard drive (*see,* Fig. 2) which may be utilized to store and retrieve software programs incorporating computer code that implements the invention, data for use with the invention and the like. Although a CD 15 is shown as an exemplary computer readable medium, other computer readable storage media including floppy disk, tape, flash memory, system memory, and hard drive may be utilized. Additionally, a data signal embodied in a carrier wave (*e.g.,* in a network including the internet) may be the computer readable storage medium.

Fig. 2 shows a system block diagram of computer system 1 used to execute the

software of an embodiment of the invention. As in Fig. 1, computer system 1 includes monitor 3, and keyboard 9, and mouse 11. Computer system 1 further includes subsystems such as a central processor 51, system memory 53, fixed storage 55 (*e.g.*, hard drive), removable storage 57 (*e.g.*, CD-ROM), display adapter 59, sound card 61, speakers 63, and network interface 65. Other computer systems suitable for use with the invention may include additional or fewer subsystems. For example, another computer system may include more than one processor 51 or a cache memory. Computer systems suitable for use with the invention may also be embedded in a measurement instrument or performed using ASIC devices or the like.

In one aspect of the invention, methods, systems and computer software products are provided to minimize the edges between features in a photo-lithograhic synthesis of polymers.

Methods of forming high density arrays of oligonucleotides, peptides and other polymer sequences with a minimal number of synthetic steps are disclosed in, for example, 5,143,854, 5,252,743, 5,384,261, 5,405,783, 5,424,186, 5,429,807, 5,445,943, 5,510,270, 5,677,195, 5,571,639, 6,040,138, all incorporated herein by reference for all purposes. The oligonucleotide analogue array can be synthesized on a solid substrate by a variety of methods, including, but not limited to, light-directed chemical coupling, and mechanically directed coupling. See Pirrung et al., U.S. Patent No. 5,143,854 (see also PCT Application No. WO 90/15070) and Fodor *et al.*, PCT Publication Nos. WO 92/10092 and WO 93/09668 and U.S. Pat. No. 5,677,195 which disclose methods of forming vast arrays of peptides, oligonucleotides and other

molecules using, for example, light-directed synthesis techniques. See also, Fodor *et al.*, Science, 251, 767-77 (1991). These procedures for synthesis of polymer arrays are now referred to as VLSIPS™ procedures. Using the VLSIPS™ approach, one heterogeneous array of polymers is converted, through simultaneous coupling at a number of reaction sites, into a different heterogeneous array. See, U.S. Patent Nos. 5,384,261 and 5,677,195.

The development of VLSIPS™ technology as described in the above-noted U.S. Patent No. 5,143,854 and PCT patent publication Nos. WO 90/15070 and 92/10092, is considered pioneering technology in the fields of combinatorial synthesis and screening of combinatorial libraries.

In brief, the light-directed combinatorial synthesis of oligonucleotide arrays on a glass surface proceeds using automated phosphoramidite chemistry and chip masking techniques. In one specific implementation, a glass surface is derivatized with a silane reagent containing a functional group, e.g., a hydroxyl or amine group blocked by a photolabile protecting group. Photolysis through a photolithogaphic mask is used selectively to expose functional groups which are then ready to react with incoming 5'-photoprotected nucleoside phosphoramidites. The phosphoramidites react only with those sites which are illuminated (and thus exposed by removal of the photolabile blocking group). Thus, the phosphoramidites only add to those areas selectively exposed from the preceding step. These steps are repeated until the desired array of sequences have been synthesized on the solid surface. Combinatorial synthesis of different oligonucleotide analogues at different locations

on the array is determined by the pattern of illumination during synthesis and the order of addition of coupling reagents.

In the event that an oligonucleotide analogue with a polyamide backbone is used in the VLSIPS™ procedure, it is generally inappropriate to use phosphoramidite chemistry to perform the synthetic steps, since the monomers do not attach to one another via a phosphate linkage. Instead, peptide synthetic methods are substituted. *See, e.g.,* Pirrung *et al.* U.S. Pat. No. 5,143,854.

Peptide nucleic acids are commercially available from, *e.g.*, Biosearch, Inc. (Bedford, MA) which comprise a polyamide backbone and the bases found in naturally occurring nucleosides. Peptide nucleic acids are capable of binding to nucleic acids with high specificity, and are considered "oligonucleotide analogues" for purposes of this disclosure.

In addition to the foregoing, additional methods which can be used to generate an array of oligonucleotides on a single substrate are described in PCT Publication No. WO 93/09668. In the methods disclosed in the application, reagents are delivered to the substrate by either (1) flowing within a channel defined on predefined regions or (2) "spotting" on predefined regions or (3) through the use of photoresist. However, other approaches, as well as combinations of spotting and flowing, may be employed. In each instance, certain activated regions of the substrate are mechanically separated from other regions when the monomer solutions are delivered to the various reaction sites.

As described above, one method of synthesizing an oligonucleotide array or

peptide array is by a photolithographic VLSIPS™ method. In this method, light is used to direct the synthesis of oligonucleotides in an array. In each step, light is selectively allowed through a mask to expose cells in the array, activating the oligonucleotides in that cell for further analysis. For every synthesis step, there is a mask with corresponding open (allowing light) and closed (blocking light) cells. Each mask corresponds to a step of combinatorial synthesis. This method is useful for synthesizing many different types of polymers including oligonucleotides (often used as probes against nucleic acid target), peptides and polysaccharides. However, for the purpose of clarity, various aspects of the invention are described using exemplary embodiments for synthesizing oligonucleotide probes.

As used herein, edges are the differences between polymer synthesis sites. In some embodiments, edges are difference between the synthesis steps used for one probe and the synthesis steps used for another probe. Due to reflection, internal reflection, scattering and other effects during photodirected synthesis, light does not precisely fill the areas designed to be illuminated. Light often leaks from these areas into nearby regions. Every edge is a possibility for light leakage, which may lead to a lower quality set of probes being synthesized. It is desirable to minimize such unintended illumination.

Edge counts may be integers: zero, one, or any other number. Because light leakage may occur over long distances (60 microns), in some instances it may be desirable to obtain a weighted edge count (WEIGHTED EDGE COUNT) taking into account the distance to the cell leaking light. For example, if the light

leakage halves every 10 microns, and features are 20 microns across, then it is reasonable to weight the edges between a target cell and a cell one feature distant as 1/4 the edges of the cell immediately adjacent to the target cell.

One of skill in the art would appreciate that this is one of many possible weighting functions. Other weighing functions are also within the scope of the invention. For computational efficiency, in one embodiment, only nearby cells need to be counted, since weights for extremely distant cells are negligible.

In one aspect of the invention, methods and computer software products are provided to arrange the probes in an order such that the total edge count between probes adjacent in the order are reduced. In a synthesis scheme of N synthesis steps, each probe can be viewed as a binary vector of length N. The number of edges between two probes is the number of places where the binary vectors are different, the so called Hamming distance. If an ordered list of probes are assigned to spatial positions in such a manner that are typically probes adjacent in the list are adjacent on the chip, then the number of edges on the chip will be similar to the number of edges in the list. Thus, finding an ordering of the vectors in the list so that the total distance between all adjacent vectors is minimal will provide a reduced set of edges on the chip. In some embodiments of the invention, an ordering of the list is provided by performing travelling salesman optimization. In one embodiment, a locally greedy insertion heuristic is used to construct the ordered list.

As used herein, the term travelling salesman optimization refers to methods, steps, algorithm, solution or the like for performing optimization (particularly

minimization) that are also useful for solving the travelling salesman problem. Many well known approximate solutions, methods, steps and algorithms have been developed to perform travelling salesman problem in the art (see, e.g., David Applegate, Robert Bixby, Vasek Chvátal, and William Cook, On the solution of travelling salesman problems, Documenta Mathematica, vol. 3, pp. 645 - 656, 1998. Extra volume ICM 1998; David Applegate, Robert Bixby, Vasek Chvátal, and William Cook, Finding tours in the tsp, Tech. Rep. TR99-05, Departement of Computational and Applied Mathematics, Rice University, 1999; Leonard M. Adleman, Molecular computation of solutions to combinatorial problems, Science, vol. 266, pp. 1021 - 1024, 1994; Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel, A polyhedral study of the asymmetric travelling salesman problem with time windows. Available via WWW at tt www.zib.de, February 1997. Preprint.; Norbert Ascheuer, Matteo Fischetti, and Martin Grötschel, Solving the asymmetric travelling salesman problem with time windows by branch-and-cut, August 1999. Preprint SC 99-31; Norbert Ascheuer, Michael Jünger, and Gerhard Reinelt, A branch & cut algorithm for the asymmetric hamiltonian path problem with precedence constraints. Available via www at www.zib.de, December 1997; Edward K. Baker, An exact algorithm for the time-constrained travelling salesman problem, Operations Research, vol. 31, pp. 938 - 945, September-October 1983; Rainer E. Burkard, Vladimir G. Deineko, René van Dal, Jack A. A. van~der Veen, and Gerhard J. Woeginger, Well-solvable special cases of the TSP: A survey, Tech. Rep. 52, Karl-Franzens-Universität & Technische Universität Graz, Dezember 1995; Egon Balas

and Matteo Fischetti, A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets, Mathematical Programming, vol. 58, no. 3, pp. 325 - 352, 1993; Egon Balas, Matteo Fischetti, and William R. Pulleyblank, The precedence-constrained asymmetric traveling salesman polytope, Mathematical Programming, vol. 68, no. 3, pp. 241 - 265, 1995; Giovanni Cesari, Divide and conquer strategies for parallel TSP heuristics, Computers & Operations Research, vol. 23, no. 7, pp. 681 - 694, 1996; Harlan Crowder and Manfred W. Padberg, Solving large-scale symmetric travelling salesman problems to optimality, Management Science, vol. 26, pp. 495 - 509, March 198, all incorporated by reference herein for all purposes). These methods, solutions, and algorithm are useful for at least some embodiment of the invention to minimize the edges.

In another aspect of the invention, probes very often come in pairs or quadruplets of related probes. These related probes almost always have only one or two edges between them. Thus, it is useful to assign the related probe sets as blocks, rather than individual probes in some embodiments. As used herein, the term block may contain a single probe or related probes or probe sets.

One of skill in the art would appreciate that this is one of many possible weighting functions. Other weighing functions are also within the scope of the invention. For computational efficiency, in one embodiment, only nearby cells need to be counted, since weights for extremely distant cells are negligible.

The edge minimization problem may be solved using a computer to arrange the blocks of probes so that the edge count or weighted edge count is minimal.

Normally, there are many features on the chip that may not be moved (control probes, text, spatial normalization features), and these may form constraints on the process of minimization.

One method of solving the edge minimization problem is to use an annealing approach. In this approach, pairs of blocks of probes are swapped at random - if the random swap results in an improvement, it is always kept. If the swap increases the edge count, then the resulting arrangement is kept with a probability dependent upon a hidden variable of Temperature (the temperature is a parameter which controls the bias in optimization towards locally good solutions), otherwise the swap is undone.

Lower (cooler) temperatures reject swaps that increase the edge count more often than higher temperatures. Simulated annealing with properly cooled temperatures is an often-used tool for large optimization problems. However, annealing of arrays takes a long time in practice.

In yet another aspect of the invention, a simpler and faster algorithm employing a locally greedy approach is provided (Figure 3). A locally greedy approach considers one "slot" on an array which is a substrate containing spatially arranged polymers such as oligonucleotide probes at a time where a block of probes can be placed. A set of blocks that have not yet been optimized are tried and the optimal (normally the block with the minimal edge count) block is chosen and placed into that slot (displacing the block currently in that slot, if the slot is not empty). This process continues, considering all the slots on the array that have not yet been optimized until all slots have had a "locally best" block placed in them.

In one implementation, all blocks that are valid (i.e. are specified as allowed to be moved by the user) are removed from the array, leaving a set of empty slots to be filled. These slots are then searched in a diagonal fashion, with a user-specified number of blocks specified to search for each slot. Thus, in a two dimensional array, each block typically is compared to previously placed blocks to the "north" and "west" directions, with the "east" and "south" directions consisting of empty slots. One of skill in the art would appreciate that other direction of comparison may also be used.

For example, in one embodiment of computer implemented method, 135,000 blocks consisting of pairs of probes could be found on an expression chip. The order of the blocks is shuffled randomly (Figure 3, 302), and then the first subset of 1000 blocks (in the computer software product for performing the method, the number of blocks in the subset may be specified by a user, preferrably, the number may be in the range of 20-100, 100-500, 500-1000, 1000-10000) are checked against the first slot on the chip (305). The best fitting block (least edge count) is placed into that slot, leaving 134,499 blocks remaining (306). This process continues, moving across the chip adding to empty slots. Towards the end of the chip, when there are fewer than 1000 blocks remaining, only the actual number of blocks remaining are searched when attempting to fill an empty slot (304).

The user specified subset of blocks speeds up the computation by limiting the search to only a few blocks per slot, rather than comparing all the remaining blocks to the current empty slot. There is a cost in the amount of optimization done, but this parameter allows the user to trade off the amount of computation done against the

quality of optimization (exact trade-offs depend on the structure of the array). It is of course obvious that the order in which the empty slots are traversed is not crucial, however, experimentation has determined that diagonal replacement works well, with a possible slight advantage over horizontal or vertical replacement.

Computer software products for implementing the locally greedy optimization may contain computer codes for performing each of the steps of the computer implemented methods described above.

In an additional aspect of the invention, methods, systems and computer software products are provided for solving Robust Arrangement Problem (RAP).

Oligonucleotide arrays for monitoring gene expression (See, e.g., U.S. Patent No. 6,040,138, which is incorporated herein by reference for all for detailed description of using oligonucleotide array for gene expression monitoring) may have certain number of probe pairs (generally a probe that is designed to be complementary to a target gene and a probe that is designed to contain at least one mismatch), such as 10, 15, or 20 probe pairs devoted to any given gene. Local problems (flecks of dust, bubbles, defects) may occur on the array, and if the probe pairs are arranged adjacent to each other, there may be no informative probes remaining for that gene if a defect occurs. The RAP is a probe distribution problem of arranging all the probe pairs on the chip, so that of the N (typically, 10, 15 or 20 pairs) probe pairs associated with any given gene, no more than K, such as 2, 3, 4 or 5, of them are within a radius R of each other. While methods and computer software for solving the RAP problem is described using probe pairs as examples, the

methods and computer software is also useful for other probe arrangement. For example, mismatch probes may be unnecessary for gene expression monitoring purpose in some embodiments. In such embodiments, the RAP problem is to reduce non-robust probes rather than adjacent probe pairs.

Typically, for an edge optimized chip using the above-described methods, software or system, the probes are scrambled across the chip, and the probe pairs for a given gene are unlikely to be near each other. However, there may be some positions where K probe pairs for a given gene are within the specified radius R. As used herein, a non-robust (or bad or adjacent) probe pair is a probe pair which occurs as one of the at least K probe pairs associated with a given gene within the specified radius.

In the typical expression array, of the large number of probe-pairs on a chip (>100,000), after edge-optimization, typically fewer than 1% will be non-robust. If all non-robust probe pairs are removed from the chip as blocks, leaving empty slots behind, and an equal number of robust probe pairs are chosen randomly and also removed, and then these blocks are replaced (almost) randomly into the slots, the number of new non-robust blocks will be reduced greatly (typically again cut to 1% of the former value). This dilution procedure may be repeated until there are no non-robust blocks remaining.

Computer software products for solving RAP is also provided (part of edgeopt.cpp, Appendix B). In preferred embodiments, software products may contain both code for performing edge minimization and for solving RAP.

In one embodiment, the basic structure of the computer software for performing the optimization is described as follows (see, also, Figure 4): .ret and .cdl files are read in to describe a chip. Selected blocks of probes (atoms) are removed from the chip and placed on a stack. Empty spaces are left behind. Probes are then put back in a locally greedy fashion into the empty spaces. These steps may be repeated for many different types of blocks. The scrambled chips may then be output to a variety of files.

Appendix A is a computer program in c++ (travel.cpp) that is used to reducing or minimizing the edges between cells using travelling salesman optimization of an ordered list of polymers. The algorithm provides a general insertion heuristic.

Appendix B is a computer program in c++ (edgeopt.cpp) that operate in a locally greedy fashion to optimize the sequence chips in two dimensions. Optimizing chips in two dimensions simultaneously allows for fewer edges on all sides of the probes (more optimization is possible) and for the optimization to be more uniform on all edges of the probes.

Valid commands for Edge Optimization using this exemplary software embodiment are:

lu = lower unit number of range

uu = upper unit number of range

v = value of validflag (1=valid for stripping, 0= don't move)

d = destype

h = height of block/atom (i.e. 2, 4, ...)

sl = searchlimit = max number of possibilities to search through

r = radius

m = max allowed

1. Must be first two commands given:

READCDL: in.cdl = read in cdl file

READRET: in.ret = read in ret file

2. Set valid entities for moving:

SETVALIDUNITS: lu uu v

SETVALIDAREA: x y tx ty v

SETVALIDANTIAREA: x y tx ty v

SETVALIDDESTYPE: d

3. Actually put movable blocks onto the stack:

STRIPBLOCKS: h

4. Replace blocks into the allowed space:

DIAGONALREPLACEMENT: sl

HORIZONTALREPLACEMENT: sl

AGGREPLACEMENT: sl

5. Do proximity checking, and fix bad (adjacent) entities:

SETPROXIMITY: r m

FIXBAD: sl

Steps 2-5 may be repeated as needed to optimize different sets of blocks on the chip.

6. Output the data:

DUMPCDL: out.cdl

DUMPRET: out.ret

DUMPMUT: out.mut

DUMPDIFF: out.dff

7. Exit gracefully:

END:

While the edge minimization methods and software products are described for use in the synthesis of oligonucleotide arrays using VLSIP™ technology employing masks, the method and software products of the invention are also useful for many other purposes including maskless synthesis. For example, the methods and software are useful for VLSIP™ technology employing micro-mirrors instead of masks (U.S. Patent Application Serial Number 09/318,775, *see also*, Signh-Gasson et al., Maskless fabrication of light-directed oligonucleotide microarrays using a digital micromirror array, Nature-Biotechnology 17:974-978, 1999, both incorporated herein by reference for all purposes). It would also be apparent to those with skill in the art that the methods and software products of the invention is also useful for the synthesis

of sequence arrays using ink-jet printing or mechanic flow control. More generally, the methods and software products of the invention are useful for the minimization of edges between features.

The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. Merely by way of example, while the invention is illustrated with particular reference to the evaluation of DNA, the methods can be used in the synthesis and data collection from chips with other materials synthesized thereon, such as RNA and peptides (natural and unnatural). The scope of the invention should, therefore, be determined not with reference to the above description, but instead should be determined with reference to the appended claims along with their full scope of equivalents.

We claim:

1. A computer implemented method for arranging polymers for combinatorial synthesis of said polymers on a substrate comprising:

reducing edge count between said polymers comprising computer-implemented steps for optimization of an ordered list of polymers.

2. The method of Claim 1 wherein said steps for optimization comprises steps for travelling salesman optimization of said ordered list of polymers.

3. The method of Claim 2 wherein said travelling salesman optimization is performed by means of a locally greedy insertion heuristic.

4. A computer implemented method for arranging polymers for combinatorial synthesis of said polymers on a substrate comprising:

reducing edge count between said polymers comprising:

dividing said polymers into a plurality of blocks, wherein each of said block comprising one or more related polymers, wherein each of said blocks is to be assigned to one slot on said substrate; and

selecting a subset of said blocks from unassigned blocks; and

assigning one block of said blocks in said set to an empty slot, wherein said one block is the best fitting and results in a least edge count among said blocks of said

subset.

5. The method of Claim 4 further comprising repeating said steps of selecting and assigning until all blocks are assigned.

6. The method of Claim 5 wherein said assigning comprises:

computing a plurality of edge counts, each of said edge counts represents the result of assigning one block of said subset to said empty slot;

comparing said edge counts and selecting said best fitting block, wherein said best fitting block has said least edge count.

7. The method of Claim 6 wherein said blocks are ordered randomly and said selecting step comprises selecting the first subset among unassigned blocks.

8. The method of Claim 7 wherein the last of said subsets has no more than 100 blocks and other said subset has at least 20 blocks and no more than 100 blocks.

9. The method of Claim 7 wherein the last of said subset has no more than 1000 blocks and other said subset has at least 100 blocks and no more than 1000 blocks.

10. The method of Claim 7 wherein the last of said subsets has no more than 10000 blocks and other said subset has at least 1000 blocks and no more than 10000

blocks.

11. The method of Claim 7 wherein said polymers are oligonucleotides.

12. The method of Claim 11 wherein said combinatorial synthesis is radiation directed synthesis.

13. The method of Claim 12 wherein said radiation directed synthesis comprises steps of controlling irradiation to active synthesis site using a mask.

14. The method of Claim 13 wherein said edge count is a weighted edge count taking into account distance to cell leaking radiation.

15. A computer implemented method for arranging nucleic acid probes in a nucleic acid probe array comprising:

providing an arrangement of said nucleic acid probes;

reducing non-robust probes in said arrangement, wherein said non-robust probe is a probe that occurs as one of at least two (K) probes associated with a given gene within a specified area of said array, comprising:

removing non-robust blocks and optionally removing additional blocks, wherein said non-robust blocks comprises at least one non-robust probe and leaving empty slots in said initial arrangement; and

reassigning said blocks to empty slots of said arrangement.

16. The method of Claim 15 wherein said K is at least three.

17. The method of Claim 16 wherein said K is at least four.

18. The method of Claim 17 wherein said K is at least five.

19. The method of Claim 15 wherein said removing step comprises removing said additional blocks randomly.

20. The method of Claim 19 wherein said reassigning step comprises reassigning said blocks into said empty slots randomly.

21. The method of Claim 20 further comprising repeating steps of removing and reassigning.

22. A computer software product for arranging polymers for combinatorial synthesis of said polymers on a substrate comprising:

code for reducing edge count between said polymers comprising

code for optimizating an ordered list of polymers; and

a computer readable medium for storing said code.

23. The computer software product claim 22 wherein said code for optimizing comprises code for travelling salesman optimization of said ordered list of polymers.

24. The computer software product of Claim 23 wherein said code for travelling salesman optimization comprises code for a locally greedy insertion heuristic.

25. A computer software product for arranging polymers for combinatorial synthesis of said polymers on a substrate comprising:

code for reducing edge count between said polymers comprising

code for dividing said polymers into a plurality of blocks, wherein each of said blocks comprises one or more related polymers, and wherein each of said blocks is to be assigned to one slot on said substrate; and

code for selecting a subset of said blocks from unassigned blocks; and

code for assigning one block of said blocks in said set to an empty slot, wherein said one block is the best fitting and results in a least edge count among said blocks of said subset; and

a computer readable medium for storing said code.

26. The computer software product of Claim 25 further comprising code for repeating execution of said codes of selecting and assigning until all blocks are assigned.

27. The computer software product of Claim 26 wherein said code for assigning comprises:

code for computing a plurality of edge counts, each of said edge counts represents the result of assigning one block of said subset to said empty slot; and

code for comparing said edge counts and selecting said best fitting block, wherein said best fitting block has said least edge count.

28. The computer software product of Claim 27 wherein said blocks are ordered randomly and said code for selecting comprises code for selecting the first subset among unassigned blocks.

29. The computer software product of Claim 28 wherein the last of said subsets has no more than 100 blocks and other said subset has at least 20 blocks and no more than 100 blocks.

30. The computer software product of Claim 28 wherein the last of said subset has no more than 1000 blocks and other said subset has at least 100 blocks and no more than 1000 blocks.

31. The computer software product of Claim 28 wherein the last of said subsets has no more than 10000 blocks and other said subset has at least 1000 blocks and no more than 10000 blocks.

32. The computer software product of Claim 28 further comprising code for inputting size of subsets.

33. The computer software product of Claim 28 wherein said edge count is a weighted edge count taking into account distance to cell leaking radiation.

34. A computer software product for arranging nucleic acid probes in a nucleic acid probe array comprising:

code for reducing non-robust probes in an arrangement of said probes, wherein said non-robust probe is a probe that occurs as one of at least two (K) probes associated with a given gene within a specified area of said array, comprising:

code for removing non-robust blocks and optionally additional blocks, wherein non-robust blocks comprises at least one robust probe from said arrangement and leaving empty slots in said initial arrangement;

code for reassigning said blocks to empty slots of said arrangement; and

a computer readable medium for storing said codes.

35. The computer software product of Claim 34 wherein K is at least three.

36. The computer software product of Claim 34 wherein said K is at least four.

37. The computer software product of Claim 34 wherein said K is at least five.

38. The computer software product of Claim 34 wherein said code for removing comprises code for removing said other blocks randomly.

39. The computer software product of Claim 38 wherein said code for reassigning comprises code for reassigning said blocks into said empty slots randomly.

40. The computer software product of Claim 34 further comprising code for repeating execution of said codes for removing and reassigning.

METHODS, SYSTEMS AND COMPUTER SOFTWARE FOR DESIGNING AND SYNTHESIZING SEQUENCE

ABSTRACT OF THE DISCLOSURE

Embodiments of the invention provides methods, computer software products and systems for arranging polymers during combinatorial polymer synthesis so that the border or edge between synthesis site is minimized. In one embodiment, travelling salesman algorithm is used to minimize the edges. In another embodiment, a locally greedy optimization method is provided. In addition, methods and software products are provided for solving the robust arrangement problem for multi-probe gene expression arrays.

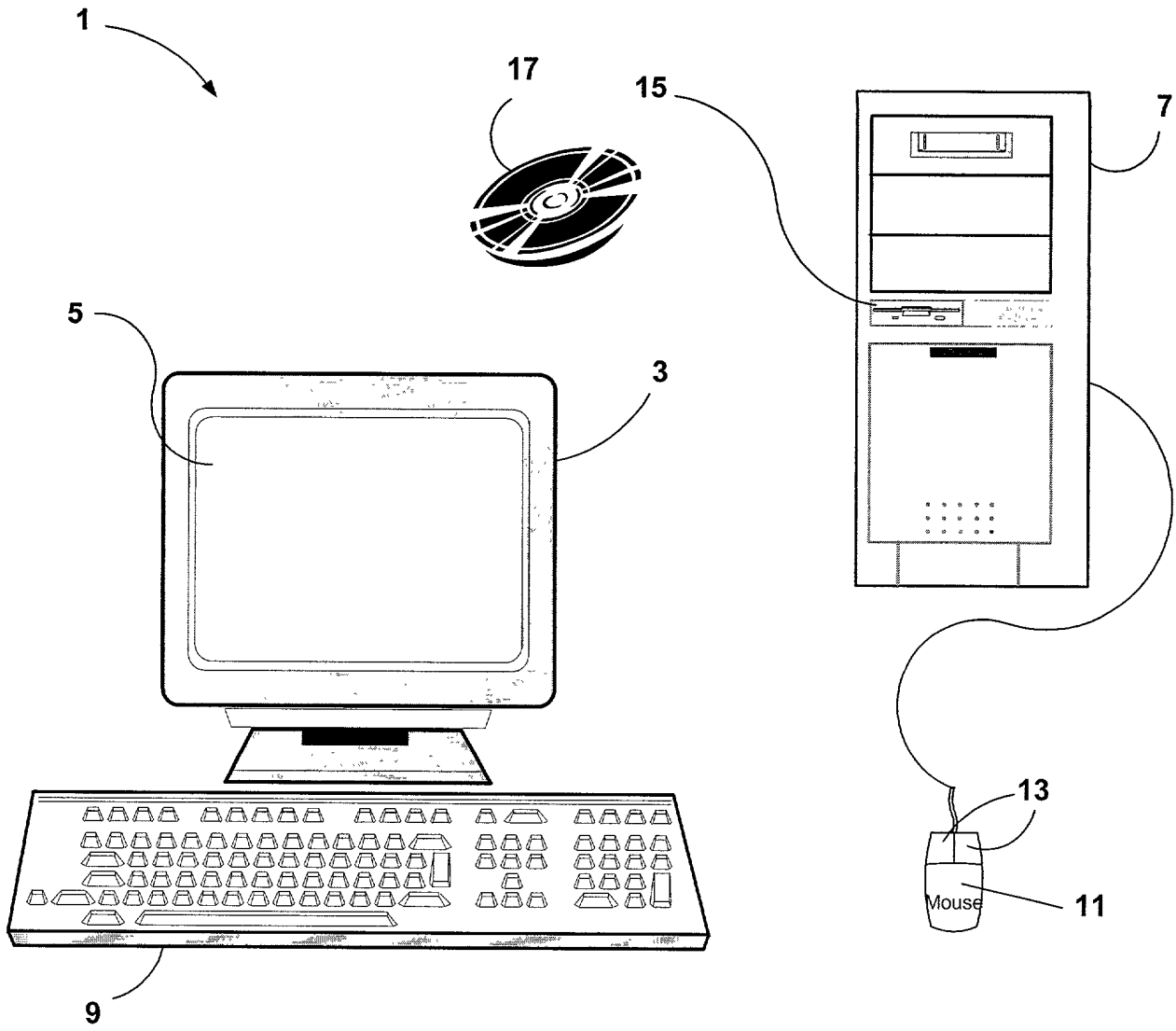

Figure 1

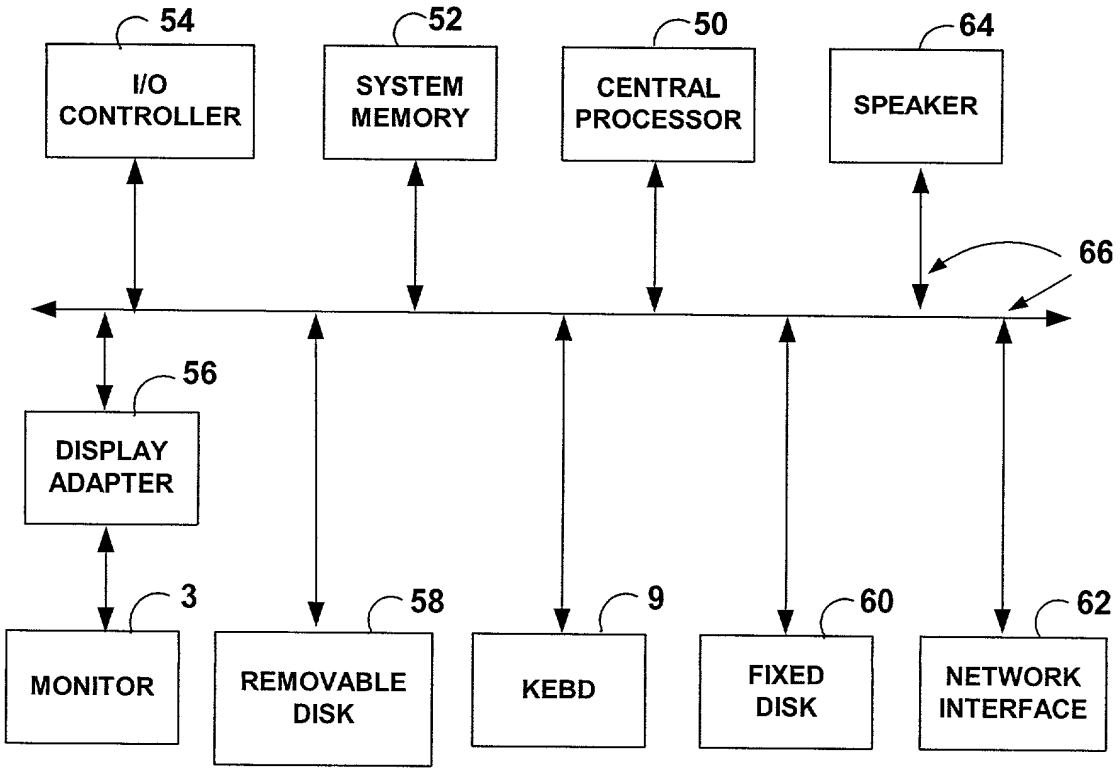
54
I/O CONTROLLER
52
SYSTEM MEMORY
50
CENTRAL PROCESSOR
64
SPEAKER
66
56
DISPLAY ADAPTER
3
MONITOR
58
REMOVABLE DISK
9
KEBD
60
FIXED DISK
62
NETWORK INTERFACE

Figure 2

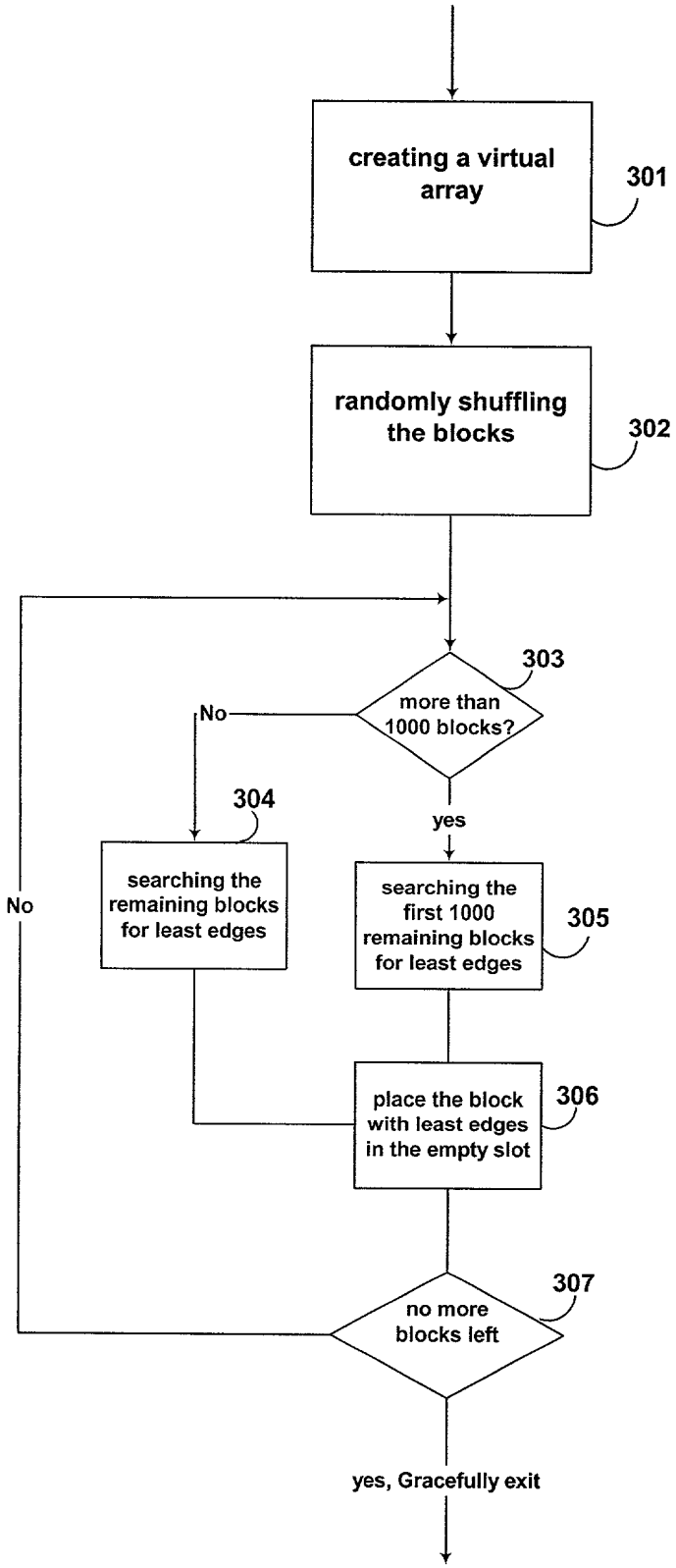
creating a virtual array
301
randomly shuffling the blocks
302
303
more than 1000 blocks?
No
yes
304
searching the remaining blocks for least edges
searching the first 1000 remaining blocks for least edges
305
No
place the block with least edges in the empty slot
306
307
no more blocks left
yes, Gracefully exit

Fig. 3

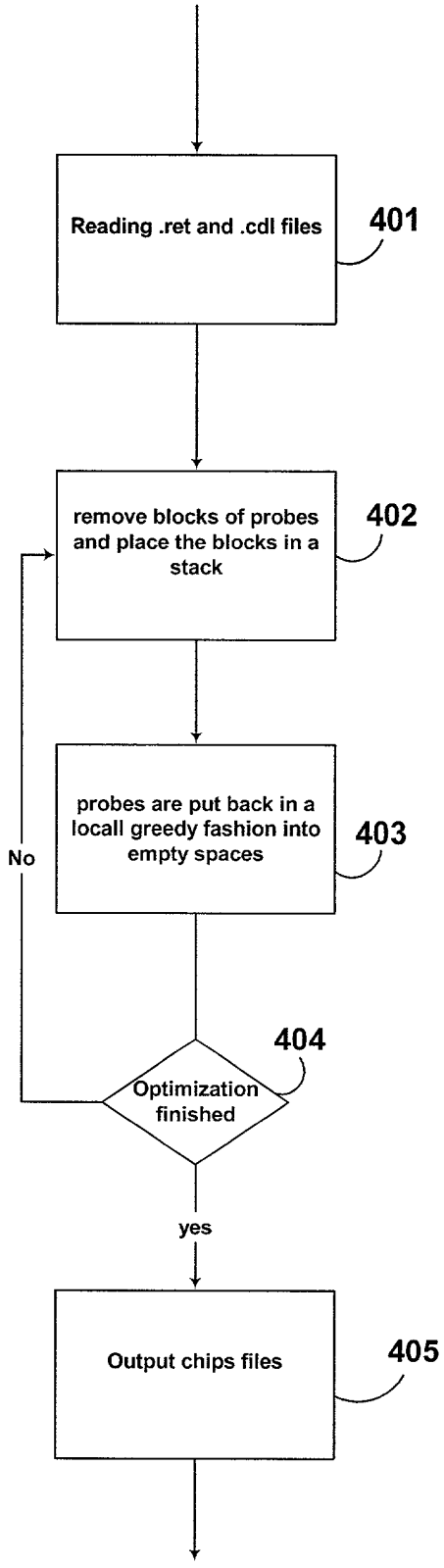
Reading .ret and .cdl files
401
remove blocks of probes and place the blocks in a stack
402
probes are put back in a locall greedy fashion into empty spaces
403
No
404
Optimization finished
yes
Output chips files
405

Fig. 4

Docket No.
3246.1

Declaration and Power of Attorney For Patent Application

English Language Declaration

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled

METHODS, SYSTEMS AND COMPUTER SOFTWARE FOR DESIGNING AND SYNTHESIZING SEQUENCE ARRAYS

the specification of which

(check one)

☒ is attached hereto.

☐ was filed on _____ as United States Application No. or PCT International Application Number _____

and was amended on _____

(if applicable)

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119(a)-(d) or Section 365(b) of any foreign application(s) for patent or inventor's certificate, or Section 365(a) of any PCT International application which designated at least one country other than the United States, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate or PCT International application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application(s)			Priority Not Claimed
_____ (Number)	_____ (Country)	_____ (Day/Month/Year Filed)	☐
_____ (Number)	_____ (Country)	_____ (Day/Month/Year Filed)	☐
_____ (Number)	_____ (Country)	_____ (Day/Month/Year Filed)	☐

I hereby claim the benefit under 35 U.S.C. Section 119(e) of any United States provisional application(s) listed below:

60/149,510	August 17, 1999
(Application Serial No.)	(Filing Date)
60/182,288	February 14, 2000
(Application Serial No.)	(Filing Date)
(Application Serial No.)	(Filing Date)

I hereby claim the benefit under 35 U. S. C. Section 120 of any United States application(s), or Section 365(c) of any PCT International application designating the United States, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT International application in the manner provided by the first paragraph of 35 U.S.C. Section 112, I acknowledge the duty to disclose to the United States Patent and Trademark Office all information known to me to be material to patentability as defined in Title 37, C. F. R., Section 1.56 which became available between the filing date of the prior application and the national or PCT International filing date of this application:

(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)
(Application Serial No.)	(Filing Date)	(Status) (patented, pending, abandoned)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

POWER OF ATTORNEY: As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith. *(list name and registration number)*

Vern Norviel, Reg. No. 32,483
Philip L. McGarrigle, Reg. No. 31,395
Wei Zhou, Reg. No. 44,419
Ellen Gonzales, Reg. No. 44,128
Alan Sherr, Reg. No. 42,147

Send Correspondence to: **Chief IP Counsel - Legal Dept.**
Affymetrix, Inc.
3380 Central Expressway
Santa Clara, CA 95051

Direct Telephone Calls to: *(name and telephone number)*
Wei Zhou (408) 731-5699

Full name of sole or first inventor **Earl A. Hubbell**
Sole or first inventor's signature Earl A. Hubbell — Date 8/15/00
Residence **416 S. Genesee, Los Angeles, CA 90036**
Citizenship **U.S.A.**
Post Office Address **Same as above**

Full name of second inventor, if any
Second inventor's signature — Date
Residence
Citizenship
Post Office Address

Appendix A

Travel.cpp

Attorney Docket 3296.1

Inventor: Earl A. Hubbell


```
#include <cstring.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define TAB '\t'
#define ARRAYHASH 256


char base_from_num(long num)
{
        switch (num%4)
        {
                case 0: return('A');
                case 1: return('C');
                case 2: return('G');
                case 3: return('T');
                default: return ('A');
        }
}

long transform(string &Test)
{
        Test.to_upper();
        for (long i=0; i<Test.length(); i++)
   {
        if (Test[i]=='A')
        Test[i]='T';
                else
              if (Test[i]=='C')
                Test[i] = 'G';
         else
                      if (Test[i]=='G')
                        Test[i]='C';
            else
                              if (Test[i]=='T')
                                              Test[i]='A';
        }
   return(TRUE);
}


class ProbeSynth{
```

```
public:
        string Probe;
        string Original;
        string Name;
        string Rename;
        long Location;
        long Unit;
        long Atom;
        // done with fancy
        char *Synth;
        long SynthLength;
        // shift by 4,8,12
        long SynthModifier;
   long MutVal; // mutation position
        ProbeSynth();
   ~ProbeSynth();
   Destroy();
        Allocate(long);
   Synthesize();
   SynthesizeFluff(long);
   SynthesizeFastFluff(long);
   SynthesizeMutant(long);
   SynthesizeMisMatch(long);
   CompSynth();
   char GetSynth(long);
        SetSynth(long, char);
   ZeroCost();
   Distance(ProbeSynth *);
        Output(ofstream &);
        List(ofstream &);
};

ProbeSynth::SynthesizeMutant(long MutateValue)
{
        long synthflag, done, fwdflag;
        long probecount, cyclecount, qloop;
   long start, finish;
        long resynth;

   start = 0;
        finish = 0;
   Allocate(4*Probe.length()+4);
        probecount = 0;
        this->MutVal = MutateValue;
        cyclecount = start;
        done = FALSE;
        synthflag = FALSE;
        fwdflag = TRUE;

        while (!done)
        {
                if (probecount==MutateValue)
                {
                        if (probecount<Probe.length()-1)
                        {
                                resynth = cyclecount;
                                cyclecount=resynth+5; /*leave room for mutant*/
                                /*synthesize mutant*/
                                synthflag = FALSE;
                                for (qloop=resynth; qloop<cyclecount-1; qloop++)
                                {
                                        if(base_from_num(qloop)==Probe[probecount])
                                        {
                        // let us know that this is a mutant (and hence different)
                   // >if< we were doing ACGT, filling in these with all 4
                   // would let us match probes well, since geographic position
                   // would matter.  But we're only doing single mismatches,
                   // for GE, so that doesn't work.
                                                SetSynth(qloop, '*');
                                                synthflag = TRUE;
                                        }
```

```
                                else
                        SetSynth(qloop, '*');
                                }
                                if (!synthflag)
                                        done = TRUE;
                                probecount++;
                cyclecount--;
                                /*cancel out cyclecount++ later*/
                        }
                        else
                        {
                                if (base_from_num(cyclecount)== Probe[probecount])
                                {
                                        SetSynth(cyclecount, base_from_num(cyclecount));
                                        synthflag = TRUE;
                                        probecount++;
                                }
                        }
                        if (probecount>Probe.length()-1)
                                done = TRUE;

                        if (finish>0 && synthflag)
                        {
                                fwdflag = FALSE;
                                probecount = Probe.length()-1;
                                cyclecount = finish;
                        }
                        else
                        {
                  cyclecount++;
                        }
                  }
                  else
                        if (base_from_num(cyclecount)==Probe[probecount])
                        {
                                if (fwdflag)
                                {
                                        SetSynth(cyclecount, base_from_num(cyclecount));
                                        probecount++;
                                        cyclecount++;
                                        if (probecount>=Probe.length())
                                                done = TRUE;
                                }
                                else
                                {
                                        SetSynth(cyclecount, base_from_num(cyclecount));
                                        probecount--;
                  cyclecount--;
                                        if (probecount<=MutateValue)
                                                done = TRUE;
                                }
                        }
                        else
                        {
                                if (fwdflag)
                  cyclecount++;
                                else
                  cyclecount--;
                        }
        }
        this->SynthLength = cyclecount; // shorten 'search' space
        return(TRUE);
}

ProbeSynth::SynthesizeMisMatch(long MutateValue)
{
        long synthflag, done, fwdflag;
        long probecount, cyclecount, qloop;
        long start, finish;
        long resynth;
```

```
    start = 0;
    finish = 0;
        Allocate(4*Probe.length()+4);
        probecount = 0;
        cyclecount = start;
    this->MutVal = MutateValue;
        done = FALSE;
        synthflag = FALSE;
        fwdflag = TRUE;

        while (!done)
        {
                if (probecount==MutateValue)
                {
                        if (probecount<Probe.length()-1)
                        {
                                resynth = cyclecount;
                                cyclecount=resynth+5; /*leave room for mutant*/
                                /*synthesize mutant*/
                                synthflag = FALSE;
                                for (qloop=resynth; qloop<cyclecount-1; qloop++)
            {
                                        if(base_from_num(qloop)==Probe[probecount])
                                        {
                                                // let us know that this is a mutant (and hence
different)
                                                SetSynth(qloop, '#');
                                                synthflag = TRUE;
                                        }
            }
                                if (!synthflag)
                                        done = TRUE;
                                probecount++;
            cyclecount--;
                                /*cancel out cyclecount++ later*/
                        }
                        else
                        {
                                if (base_from_num(cyclecount)== Probe[probecount])
                                {
                                        SetSynth(cyclecount, base_from_num(cyclecount));
                                        synthflag = TRUE;
                                        probecount++;
                                }
                        }
                        if (probecount>Probe.length()-1)
                                done = TRUE;

                        if (finish>0 && synthflag)
                        {
                                fwdflag = FALSE;
                                probecount = Probe.length()-1;
                                cyclecount = finish;
                        }
                        else
                        {
                                cyclecount++;
                        }
                   }
                   else
                        if (base_from_num(cyclecount)==Probe[probecount])
                        {
                                if (fwdflag)
                                {
                                        SetSynth(cyclecount, base_from_num(cyclecount));
                                        probecount++;
               cyclecount++;
                                        if (probecount>=Probe.length())
                                                done = TRUE;
                                }
                                else
```

```
                                {
                                        SetSynth(cyclecount, base_from_num(cyclecount));
                                        probecount--;
                        cyclecount--;
                                        if (probecount<=MutateValue)
                                                done = TRUE;
                                }
                        }
                        else
                        {
                                if (fwdflag)
                        cyclecount++;
                                else
                                        cyclecount--;
                        }
        }
        this->SynthLength = cyclecount; // shorten 'search' space
        return(TRUE);
}

char
ProbeSynth::GetSynth(long Which)
{
        if (Which < SynthLength && Which>-1)
                return(Synth[Which]);
        else
                return('.');
}

ProbeSynth::SetSynth(long Which, char What)
{
        if (Which>-1 && Which < SynthLength)
        Synth[Which] = What;
   return(TRUE);
}

ProbeSynth::ZeroCost()
{
        return(Probe.length());
}

ProbeSynth::Distance(ProbeSynth *Destination)
{
        static char testchar;
        static long count;
        static long minlen;
        static long i;

        count = 0;
        minlen = min(Destination->SynthLength, this->SynthLength);

        for (i=0; i<minlen; i++)
        {
                testchar = Destination->Synth[i];  // can guarantee i>0
                if (testchar!='.')
                        if (this->Synth[i]!=testchar)
                        {
                                if (testchar == '#')
                                        count+=2;  // heavily penalize mutants
                                else
                                        count++;
                        }
        }
        for (; i<Destination->SynthLength; i++)
        {
                testchar = Destination->Synth[i];  // can guarantee i>0
                if (testchar!='.')
                {
                        if (testchar == '#')
                                count+=2;  // heavily penalize mutants
                        else
```

```
                                count++;
                }
        }
        return(count);
}

ProbeSynth::Synthesize()
{
        long Size = CompSynth();
        Allocate(Size);
        long length, probeloop, synthloop;
        char testchar;

        length = Probe.length();
        probeloop = 0;
        synthloop = 0;
        while (probeloop<length)
        {
                testchar = base_from_num(synthloop);
                if (testchar==Probe[probeloop])
                {
                        Synth[synthloop] = Probe[probeloop];
                        probeloop++;
                }
                else
                        Synth[synthloop] = '.';
                synthloop++;
        }
        return(TRUE);
}

ProbeSynth::SynthesizeFluff(long MutateValue)
{
        long length, probeloop, synthloop, testloop;
        char testchar;

        length = Probe.length();
        long Size = 4*length+4;
        Allocate(Size);
   this->MutVal = MutateValue;
        probeloop = 0;
        synthloop = 0;
   for (probeloop=0; probeloop<length; probeloop++)
        {
                synthloop= probeloop*4;
      testchar = Probe[probeloop];
        for (testloop = 0; testloop<4; testloop++, synthloop++)
                {
                        if (base_from_num(synthloop)==testchar)
         {
                if (probeloop!=MutateValue)
                                        Synth[synthloop] = testchar;
                                else
                Synth[synthloop] = '#';
         }
                        else
                Synth[synthloop] = '.';
      }
        }
        return(TRUE);
}

ProbeSynth::SynthesizeFastFluff(long MutateValue)
{
        long length, probeloop, synthloop, testloop;
        char testchar;

        length = Probe.length();
        long Size = length+1;
        Allocate(Size);
   this->MutVal = MutateValue;
```

```
        probeloop = 0;
        synthloop = 0;
    for (probeloop=0, synthloop = 0; probeloop<length; probeloop++, synthloop++)
        {
                Synth[synthloop] = Probe[probeloop];
       if (probeloop==MutateValue)
       {
        synthloop++; // account for mutant being twice as bad
         Synth[synthloop]=Probe[probeloop];
                }
        }
        return(TRUE);
}

ProbeSynth::CompSynth()
{
        long length, probeloop, synthloop;
        char testchar;

        length = Probe.length();
        probeloop = 0;
        synthloop = 0;
        while (probeloop<length)
        {
                testchar = base_from_num(synthloop);
                if (testchar==Probe[probeloop])
                {
                        probeloop++;
                }
                synthloop++;
        }

        return(synthloop);
}


ProbeSynth::ProbeSynth()
{
        Probe = "";
   Name = "None";
        Location = 0;
   Synth = NULL;
        SynthLength = 0;
        SynthModifier = 0;
}

ProbeSynth::Allocate(long Size)
{
        Destroy();
   Synth = new char [Size];
   for (long i=0; i<Size; i++)
                Synth[i] = '.';
   SynthLength = Size;
}

ProbeSynth::Destroy()
{
        if (Synth!=NULL)
   {
        delete[] Synth;
      Synth = NULL;
      SynthLength = 0;
   }
}

ProbeSynth::~ProbeSynth()
{
        Destroy();
}

ProbeSynth::Output(ofstream &OutStream)
```

```
{
        OutStream << this->Location << TAB;
        OutStream << this->Original << TAB;
        OutStream << this->Name << TAB;
        OutStream << this->Rename << TAB;
        OutStream << this->Unit << TAB;
        OutStream << this->Atom << endl;
}

ProbeSynth::List(ofstream &OutStream)
{
        OutStream << this->Original << endl;
}

class ProbeNode{
public:
        ProbeSynth *DataPointer;
   ProbeNode *Previous;
   ProbeNode *Next;
        ProbeNode **PClosest;
        long plength;
        long marker;
        long NextCost;
        PointToData(ProbeSynth *);
        ProbeNode();
   ~ProbeNode();
   Destroy();
   DestroyPClosest();
        DestroyData();
   AllocatePClosest(long);
        LinkPrevious(ProbeNode *);
   LinkNext(ProbeNode *);
   InsertNext(ProbeNode *);
   Initialize();
        ZeroCost();
   Copy(ProbeNode *);
   Distance(ProbeNode *);
};

ProbeNode::Copy(ProbeNode *Original)
{
        if (Original!=NULL)
                DataPointer = Original->DataPointer;
        else
                return(FALSE);
   // note - do not copy Pclosest, plength - don't apply to copies
   // do not copy previous & next, because they won't correspond.
}


ProbeNode::PointToData(ProbeSynth *Data)
{
        DataPointer = Data;
}

ProbeNode::ZeroCost()
{
        return(DataPointer->ZeroCost());
}

ProbeNode::Distance(ProbeNode *Destination)
{
        return(DataPointer->Distance(Destination->DataPointer));
}

ProbeNode::ProbeNode()
{
        DataPointer = NULL;
   Previous = NULL;
   Next = NULL;
        PClosest = NULL;
```

```
    marker = 0;
        plength =0;
    NextCost = 0;
}

ProbeNode::Destroy()
{
        DataPointer = NULL;
    Previous = NULL;
    Next = NULL;
}

ProbeNode::DestroyPClosest()
{
        if (PClosest !=NULL)
        {
        delete[] PClosest;
      PClosest = NULL;
      plength = 0;
        }
}

ProbeNode::DestroyData()
{
        // dangerous - must be last called
    if (DataPointer!=NULL)
    {
                delete[] DataPointer;
       DataPointer = NULL;
        }
}

ProbeNode::AllocatePClosest(long Size)
{
        DestroyPClosest();
        PClosest = new ProbeNode * [Size];
    for (long i=0; i<Size; i++)
        PClosest[i] = NULL;
    return(TRUE);
}

ProbeNode::~ProbeNode()
{
        Destroy();
}

ProbeNode::LinkPrevious(ProbeNode *Link)
{
        Previous = Link;
}

ProbeNode::LinkNext(ProbeNode *Link)
{
        Next = Link;
}

ProbeNode::InsertNext(ProbeNode *NewNode)
{
        ProbeNode *Link;

    Link = Next;
        Next = NewNode;
    NewNode->Previous = Link->Previous;
    Next->Previous = NewNode;
    NewNode->Next = Link;
}

ProbeNode::Initialize()
{
        Previous = this;
    Next = this;
```

```
}

class TourClass{
public:
        ProbeNode *DataList;
   long Cost;
        TourClass();
        InitializeTour(ProbeNode *);
        DeleteCurrent();
        UnlinkCurrent();
        DestroyList();
        InsertAfterCurrent(ProbeNode *);
        long TestBasicInsertion(ProbeNode *);
        Rotate();
        Duplicate(TourClass &);
        InsertLeastCost(ProbeNode *);
        InsertLeastCostFromPool(TourClass &);
        InsertLeastCostByMarker(TourClass &);
        InsertLeastCostWithModifier(ProbeNode *, long);
        QuickInsertLeastCost(ProbeNode *, long);
        ~TourClass();
        Output(ofstream &);
        List(ofstream &);
};

TourClass::TourClass()
{
        DataList = NULL;
        Cost = 0;
}

TourClass::Output(ofstream &OutStream)
{
        ProbeNode *Start = this->DataList;

        this->DataList->DataPointer->Output(OutStream);
        Rotate();
        while (Start != this->DataList)
        {
                this->DataList->DataPointer->Output(OutStream);
                Rotate();
        }
}

TourClass::List(ofstream &OutStream)
{
        ProbeNode *Start = this->DataList;

        this->DataList->DataPointer->List(OutStream);
        Rotate();
        while (Start != this->DataList)
        {
                this->DataList->DataPointer->List(OutStream);
                Rotate();
        }
}

TourClass::Duplicate(TourClass &TestTour)
{
        ProbeNode *Duplicate;
        ProbeNode *Start;

   this->DestroyList();
        Start = TestTour.DataList;
   Duplicate = new ProbeNode;
   Duplicate->Copy(TestTour.DataList);
        InitializeTour(Duplicate);
   TestTour.Rotate();

   while (Start!=TestTour.DataList)
```

```
        {
         Duplicate = new ProbeNode;
       Duplicate->Copy(TestTour.DataList);
       InsertAfterCurrent(Duplicate);
                TestTour.Rotate();
       Rotate();
         }
   return(TRUE);
}

TourClass::InsertAfterCurrent(ProbeNode *NewNode)
{
        ProbeNode *Link;

        if (DataList==NULL)
        {
                InitializeTour(NewNode);
                return(TRUE);
        }
        Link = DataList->Next;
        DataList->Next = NewNode;
        Link->Previous = NewNode;
        NewNode->Next = Link;
        NewNode->Previous = DataList;
        Cost = Cost - DataList->NextCost;
        DataList->NextCost = DataList->Distance(NewNode);
        Cost = Cost + DataList->NextCost;
        NewNode->NextCost = NewNode->Distance(Link);
        Cost = Cost + NewNode->NextCost;
}

long
TourClass::TestBasicInsertion(ProbeNode *NewNode)
{
        long TestCost;

        TestCost = Cost - DataList->NextCost;
        TestCost += DataList->Distance(NewNode);
   TestCost += NewNode->Distance(DataList->Next);
   return(TestCost);
}

TourClass::Rotate()
{
        DataList = DataList->Next;
}

TourClass::InsertLeastCost(ProbeNode *NewNode)
{
        ProbeNode *Start;
        ProbeNode *BestPlace;
   long TestCost, BestCost;

   Start = DataList;
        BestPlace = DataList;
   TestCost = TestBasicInsertion(NewNode);
        BestCost = TestCost;
        Rotate();
   while (DataList!=Start)
        {
                TestCost = TestBasicInsertion(NewNode);
      if (TestCost<BestCost)
      {
       BestCost = TestCost;
         BestPlace = DataList;
      }
      Rotate();
   }
        DataList = BestPlace;
   InsertAfterCurrent(NewNode);
}
```

```
TourClass::InsertLeastCostByMarker(TourClass &Source)
{
        // searches Source for undone ones,
   // adds the closest one to the new tour
        ProbeNode *Start;
   ProbeNode *BestPlace, *BestAdd;
   ProbeNode *NewNode, *Done;
        long TestCost, BestCost;

        Start = DataList;
        BestPlace = DataList;
   NewNode = Source.DataList;
        Done = Source.DataList;
   TestCost = TestBasicInsertion(NewNode);
   BestCost = TestCost;
   Rotate();
        Source.Rotate();
   long totalmarker = 0;
        while(Source.DataList!=Done);
   {
                NewNode = Source.DataList;
           while (DataList!=Start && !NewNode->marker)
                {
                        TestCost = TestBasicInsertion(NewNode);
                        if (TestCost<BestCost)
               {
                                BestAdd = NewNode;
                 BestCost = TestCost;
                   BestPlace = DataList;
                        }
                        Rotate();
                        totalmarker = 1;
           }
      Source.Rotate();
   }
   NewNode = new ProbeNode;
   NewNode->Copy(BestAdd);
   BestAdd->marker = 1;
        DataList = BestPlace;
   InsertAfterCurrent(NewNode);
        return(totalmarker);
}

TourClass::InsertLeastCostFromPool(TourClass &Source)
{
        // searches Source for undone ones,
   // adds the closest one to the new tour
        ProbeNode *Start;
   ProbeNode *BestPlace, *BestAdd;
   ProbeNode *NewNode, *Done;
   long TestCost, BestCost;

        BestAdd = NULL;
        BestPlace = NULL;
        NewNode = Source.DataList;
        Done = Source.DataList;
        TestCost = 1000;
        BestCost = TestCost;
        Source.Rotate();
        long totalmarker = 0;
        while(Source.DataList!=Done)
        {
                NewNode = Source.DataList;
                //cout << NewNode->DataPointer->Probe << TAB << NewNode->marker << endl;
                if (!NewNode->marker)
                {
                        TestCost = TestBasicInsertion(NewNode);
                        if (TestCost<BestCost)
                        {
                                BestAdd = NewNode;
```

```
                                BestCost = TestCost;
                        }
                        totalmarker = 1;
                }
                Source.Rotate();
        }
        if (totalmarker && BestAdd)
        {
                NewNode = new ProbeNode;
                NewNode->Copy(BestAdd);
                BestAdd->marker = 1;
                InsertAfterCurrent(NewNode);
                cout << NewNode->DataPointer->Probe << endl;
                Rotate(); // go to NewNode as the favorite
        }
        return(totalmarker);
}

TourClass::InsertLeastCostWithModifier(ProbeNode *NewNode, long ModMax)
{
        ProbeNode *Start;
        ProbeNode *BestPlace;
        long BestModifier;
        long TestCost, BestCost;

        Start = DataList;
        BestPlace = DataList;
        TestCost = TestBasicInsertion(NewNode);
        BestCost = TestCost;
        BestModifier = 0;

        for (long ModLoop = 0; ModLoop<ModMax; ModLoop++)
        {
                NewNode->DataPointer->SynthModifier = ModLoop;
                Start = DataList;
                Rotate();
                while (DataList!=Start)
                {
                        TestCost = TestBasicInsertion(NewNode);
                        if (TestCost<BestCost)
                        {
                                BestCost = TestCost;
                                BestPlace = DataList;
                                BestModifier = ModLoop;
                        }
                        Rotate();
                }
        }
        DataList = BestPlace;
        NewNode->DataPointer->SynthModifier = BestModifier;
        InsertAfterCurrent(NewNode);
}

TourClass::QuickInsertLeastCost(ProbeNode *NewNode, long SearchLevel)
{
        ProbeNode *Start;
        ProbeNode *BestPlace;
        long TestCost, BestCost;

        Start = DataList;
        BestPlace = DataList;
        TestCost = TestBasicInsertion(NewNode);
        BestCost = TestCost;

        Start = DataList;
        Rotate();
        long counter = 0;
        NewNode->DataPointer->SynthModifier = 0;
        while (DataList!=Start && counter<SearchLevel)
        {
                TestCost = TestBasicInsertion(NewNode);
```

```
                if (TestCost<BestCost)
                {
                        BestCost = TestCost;
                        BestPlace = DataList;
                }
                Rotate();
                counter++;
        }
        DataList = BestPlace;
        InsertAfterCurrent(NewNode);
}

TourClass::DeleteCurrent()
{
        ProbeNode *Link;
        ProbeNode *Del;

        Link = DataList->Next;
   Del = DataList;
        Cost = Cost - DataList->NextCost;
        Cost = Cost - DataList->Previous->NextCost;
   if (Link==DataList)
   {
        DataList = NULL;
      delete Del;
                Cost = 0;
      return(TRUE);
        }
   Link->Previous = DataList->Previous;
   Link->Previous->Next = DataList->Next;
   DataList = Link;
        delete Del;
   Link->Previous->NextCost = Link->Previous->Distance(Link);
   Cost = Cost + Link->Previous->NextCost;
   return(TRUE);
}

TourClass::UnlinkCurrent()
{
        ProbeNode *Link;
   ProbeNode *Del;

        Link = DataList->Next;
   Del = DataList;
   Cost = Cost - DataList->NextCost;
        Cost = Cost - DataList->Previous->NextCost;
   if (Link==DataList)
   {
        DataList = NULL;
                Cost = 0;
      return(TRUE);
   }
   Link->Previous = DataList->Previous;
        Link->Previous->Next = DataList->Next;
   DataList = Link;
        Link->Previous->NextCost = Link->Previous->Distance(Link);
   Cost = Cost + Link->Previous->NextCost;
        return(TRUE);
}

TourClass::InitializeTour(ProbeNode *Test)
{
        DestroyList();
        DataList = Test;
        Test->Initialize();
        Cost = Test->ZeroCost();
        Test->NextCost = 0;
        return(TRUE);
}

TourClass::DestroyList()
```

```
{
        while (DataList!=NULL)
                DeleteCurrent();
}


TourClass::~TourClass()
{
        DestroyList();
}

class TSPClass{
public:
        TourClass DataSet;
        TourClass BestTour;
        TourClass CurrentTour;
        TSPClass();
        ~TSPClass();
        LoadData(string);
        LoadMutantData(string, long);
        LoadMisMatchData(string, long);
        LoadMisMatchFluffData(string, long);
        LoadMisMatchFastFluffData(string, long);
        LoadExpressionData(string, long);
        LoadSingleExpressionData(string);
        LoadExpressionDataByUnit(string, long, long);
        LoadExpressionFluffData(string, long);
        LoadChipData(string);
        GenerateTourByInsertion();
        GenerateTourByInsertionAndDeletion();
        GenerateTourByClosestInsertion();
        GenerateTourByClosestPool();
        GenerateTourByInsertionWithModifier(long);
        GenerateQuickTourByInsertion(long);
        ImproveTourByReplacement(long);
        OutputTour(string);
        AppendTour(string);
        ListTour(string);
        DestroyData();
//   Geni(int);
};

TSPClass::TSPClass()
{
}

TSPClass::DestroyData()
{
        ProbeNode *DataStart;

        DataStart = DataSet.DataList;
        DataSet.DataList->DestroyData();
        DataSet.Rotate();

   while(DataStart!=DataSet.DataList)
   {
          DataSet.DataList->DestroyData();
               DataSet.Rotate();
   }
        return(TRUE);
}

TSPClass::~TSPClass()
{
        //DestroyData();
}

TSPClass::GenerateTourByInsertion()
{
        ProbeNode *DataStart;
        ProbeNode *TempData;
```

```
        DataStart = DataSet.DataList;
        TempData = new ProbeNode;
        TempData->Copy(DataSet.DataList);
        BestTour.InitializeTour(TempData);
        DataSet.Rotate();

        long counter = 0;
        while(DataStart!=DataSet.DataList)
        {
                TempData = new ProbeNode;
                TempData->Copy(DataSet.DataList);
                //cout << DataSet.DataList->DataPointer->Probe << TAB << counter << TAB;
                BestTour.InsertLeastCost(TempData);
                //cout << BestTour.Cost << endl;
                DataSet.Rotate();
                if (counter%100==0)
                        cout << counter << endl;
                counter++;
        }
        return(TRUE);
}


TSPClass::GenerateTourByInsertionAndDeletion()
{
        ProbeNode *DataStart;
        ProbeNode *TempData;


        DataStart = DataSet.DataList;
        if (DataSet.DataList->marker>0)
                DataSet.Rotate();
        while (DataSet.DataList->marker>0 && DataSet.DataList!=DataStart)
        {
                DataSet.Rotate();
        }
        if (DataSet.DataList->marker>0)
                return(FALSE);
        DataStart = DataSet.DataList;
        DataStart->marker = 1;

        TempData = new ProbeNode;
        TempData->Copy(DataSet.DataList);
        BestTour.InitializeTour(TempData);

        long Unit = TempData->DataPointer->Unit;

        DataSet.Rotate();

        long counter = 0;
        while(DataStart!=DataSet.DataList)
        {
                if (DataSet.DataList->DataPointer->Unit==Unit && DataSet.DataList->marker<1)
                {
                        TempData = new ProbeNode;
                        TempData->Copy(DataSet.DataList);
                        cout << DataSet.DataList->DataPointer->Name << TAB << DataSet.DataList-
>DataPointer->Unit << TAB << counter << endl;
                        BestTour.InsertLeastCost(TempData);
                        DataSet.DataList->marker = 1;
                //cout << BestTour.Cost << endl;
                }
                DataSet.Rotate();
                counter++;
                if (counter%1000==0)
                        cout << counter << endl;
        }
        return(TRUE);
}
```

```
TSPClass::GenerateTourByClosestInsertion()
{
        ProbeNode *DataStart;
   ProbeNode *TempData;

        DataStart = DataSet.DataList;
   TempData = new ProbeNode;
        TempData->Copy(DataSet.DataList);
   DataSet.DataList->marker = 1; // set on this course
        BestTour.InitializeTour(TempData);

        long notdone = 1;
        long counter = 0;
   while (notdone)
        {
                notdone = BestTour.InsertLeastCostByMarker(DataSet);
      if (counter%100==0)
              cout << counter << endl;
                counter++;
   }
        return(TRUE);
}

TSPClass::GenerateTourByClosestPool()
{
        ProbeNode *DataStart;
        ProbeNode *TempData;

        DataStart = DataSet.DataList;
        TempData = new ProbeNode;
        TempData->Copy(DataSet.DataList);
        DataSet.DataList->marker = 1; // set on this course
        BestTour.InitializeTour(TempData);

        long notdone = 1;
        long counter = 0;
        while (notdone)
        {
                notdone = BestTour.InsertLeastCostFromPool(DataSet);
                counter++;
        }
        return(TRUE);
}

TSPClass::GenerateTourByInsertionWithModifier(long ModMax)
{
        ProbeNode *DataStart;
        ProbeNode *TempData;

        DataStart = DataSet.DataList;
        TempData = new ProbeNode;
        TempData->Copy(DataSet.DataList);
        BestTour.InitializeTour(TempData);
        DataSet.Rotate();

        long counter = 0;
        while(DataStart!=DataSet.DataList)
        {
                TempData = new ProbeNode;
                TempData->Copy(DataSet.DataList);
                BestTour.InsertLeastCostWithModifier(TempData, ModMax);
                if (counter%100==0)
                {
                        cout << DataSet.DataList->DataPointer->Probe << TAB << counter << TAB;
                        cout << BestTour.Cost << endl;
                }
                DataSet.Rotate();
                counter++;
        }
        return(TRUE);
}
```

```
TSPClass::GenerateQuickTourByInsertion(long SearchLevel)
{
        ProbeNode *DataStart;
        ProbeNode *TempData;

        DataStart = DataSet.DataList;
        TempData = new ProbeNode;
        TempData->Copy(DataSet.DataList);
        BestTour.InitializeTour(TempData);
        DataSet.Rotate();

        long counter = 0;
        while(DataStart!=DataSet.DataList)
        {
                TempData = new ProbeNode;
                TempData->Copy(DataSet.DataList);
                BestTour.QuickInsertLeastCost(TempData, SearchLevel);
                if (counter%100==0)
                {
                        cout << DataSet.DataList->DataPointer->Probe << TAB << counter << TAB;
                        cout << BestTour.Cost << endl;
                }
                DataSet.Rotate();
                counter++;
        }
        return(TRUE);
}

TSPClass::ImproveTourByReplacement(long ReplaceSize)
{
        ProbeNode *DataStart;
        ProbeNode *TempData;


   long counter = 0;
        while(counter<ReplaceSize)
        {
                TempData = BestTour.DataList;
                if (TempData->marker<1)
      {
                        TempData->marker = 1;
                        BestTour.UnlinkCurrent();
             cout << TempData->DataPointer->Probe << TAB << counter << TAB;
             BestTour.InsertLeastCost(TempData);
           cout << BestTour.Cost << endl;
                }
      else
                        BestTour.Rotate();
      counter++;
        }
        return(TRUE);
}

TSPClass::LoadData(string FileName)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
   string TestString;
   ProbeNode *NewNode;
        float test; // soak up number
        string junkstring;
```

```
        while (!ExpStream.eof())
        {
                ExpStream >> TestString >> junkstring >> test;
                if (TestString.length()>1)
      {
                        NewNode = new ProbeNode;
                        NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Original = TestString;
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->Synthesize();
                        this->DataSet.InsertAfterCurrent(NewNode);
                        //cout << TestString << endl;
                }
        }
        ExpStream.close();
        cout << DataSet.Cost << endl;
}

TSPClass::LoadMutantData(string FileName, long MutateValue)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
        string TestString;
        ProbeNode *NewNode;
        float test; // soak up number

        while (!ExpStream.eof())
        {
                ExpStream >> TestString >> test;
                if (TestString.length()>1)
      {
             NewNode = new ProbeNode;
          NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->SynthesizeMutant(MutateValue);
        this->DataSet.InsertAfterCurrent(NewNode);
                }
        }
        ExpStream.close();
   cout << DataSet.Cost << endl;
}

TSPClass::LoadMisMatchData(string FileName, long MutateValue)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
   string TestString;
        ProbeNode *NewNode;
        float test; // soak up number

        while (!ExpStream.eof())
        {
```

```
                ExpStream >> TestString >> test;
                if (TestString.length()>1)
                {
                        NewNode = new ProbeNode;
            NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->SynthesizeMisMatch(MutateValue);
                        this->DataSet.InsertAfterCurrent(NewNode);
        }
          }
          ExpStream.close();
    cout << DataSet.Cost << endl;
}

TSPClass::LoadMisMatchFluffData(string FileName, long MutateValue)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
    string TestString;
        ProbeNode *NewNode;
        float test; // soak up number

        while (!ExpStream.eof())
        {
                ExpStream >> TestString >> test;
       if (TestString.length()>1)
                {
              NewNode = new ProbeNode;
                        NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Probe = TestString;
           NewNode->DataPointer->SynthesizeFluff(MutateValue);
        this->DataSet.InsertAfterCurrent(NewNode);
        }
          }
          ExpStream.close();
    cout << DataSet.Cost << endl;
}

TSPClass::LoadMisMatchFastFluffData(string FileName, long MutateValue)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
    string TestString;
        ProbeNode *NewNode;
        float test; // soak up number

        while (!ExpStream.eof())
        {
                ExpStream >> TestString >> test;
       if (TestString.length()>1)
                {
              NewNode = new ProbeNode;
                        NewNode->DataPointer = new ProbeSynth;
```

```
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->SynthesizeFastFluff(MutateValue);
                        this->DataSet.InsertAfterCurrent(NewNode);
                }
                }
                ExpStream.close();
                cout << DataSet.Cost << endl;
}

TSPClass::LoadExpressionData(string FileName, long MutateValue)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
        string TestString;
        ProbeNode *NewNode;
        float test; // soak up number
        long facevalue;
        long Unit, Atom;
        string Name, Rename;
        string Junk;
        long counter = 0;
        ExpStream >> Junk >> Junk >> Junk >> Junk >> Junk >> Junk;
        while (!ExpStream.eof())
        {
                ExpStream >> facevalue >> TestString >> Name >> Rename >> Unit >> Atom;
                if (TestString.length()>1)
                {
                        NewNode = new ProbeNode;
                        NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Original = TestString;
                        transform(TestString);
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->Name = Name;
                        NewNode->DataPointer->Rename = Rename;
                        NewNode->DataPointer->Location = facevalue;
                        NewNode->DataPointer->Unit = Unit;
                        NewNode->DataPointer->Atom = Atom;
                        NewNode->DataPointer->SynthesizeMisMatch(MutateValue);
                        this->DataSet.InsertAfterCurrent(NewNode);
                }
                counter++;
                if (counter%100==0)
                        cout << counter << TAB << Name << TAB << TestString << endl;
        }
        ExpStream.close();
        cout << DataSet.Cost << endl;
}
TSPClass::LoadSingleExpressionData(string FileName)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        int probeloop = 0;
        string TestString;
        ProbeNode *NewNode;
```

```
        float test; // soak up number
        int facevalue;
        int Unit, Atom;
        string Name;
        string Junk;
        int counter = 0;
        ExpStream >> Junk >> Junk >> Junk >> Junk >> Junk >> Junk;
        while (!ExpStream.eof())
        {
                ExpStream >> facevalue >> TestString >> Name >> Junk >> Unit >> Atom;
                if (TestString.length()>1)
                {
                        NewNode = new ProbeNode;
                        NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Original = TestString;
                        transform(TestString);
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->Name = Name;
                        NewNode->DataPointer->Location = facevalue;
                        NewNode->DataPointer->Unit = Unit;
                        NewNode->DataPointer->Atom = Atom;
                        NewNode->DataPointer->Synthesize();
                        this->DataSet.InsertAfterCurrent(NewNode);
                }
                counter++;
                if (counter%100==0)
                        cout << counter << TAB << Name << TAB << TestString << endl;
        }
        ExpStream.close();
        cout << DataSet.Cost << endl;
}

TSPClass::LoadExpressionDataByUnit(string FileName, long MutateValue, long UnitLimit)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
        string TestString;
        ProbeNode *NewNode;
        float test; // soak up number
        long facevalue;
        long Unit, Atom;
        string Name;
        string Junk;
        long counter = 0;
        while (!ExpStream.eof())
        {
                ExpStream >> facevalue >> TestString >> Name >> Junk >> Unit >> Atom;
                if (TestString.length()>1 && Unit==UnitLimit)
                {
                        NewNode = new ProbeNode;
                        NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Original = TestString;
                        transform(TestString);
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->Name = Name;
                        NewNode->DataPointer->Location = facevalue;
                        NewNode->DataPointer->Unit = Unit;
                        NewNode->DataPointer->Atom = Atom;
                        NewNode->DataPointer->SynthesizeMisMatch(MutateValue);
                        this->DataSet.InsertAfterCurrent(NewNode);
                }
                counter++;
```

```
      if (counter%100==0)
        cout << counter << TAB << Name << TAB << TestString << endl;
        }
        ExpStream.close();
        cout << DataSet.Cost << endl;
}

TSPClass::LoadExpressionFluffData(string FileName, long MutateValue)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
        string TestString;
        ProbeNode *NewNode;
        float test; // soak up number
        long facevalue;
        long Unit, Atom;
        string Junk;
        string Name;
        long counter = 0;
        while (!ExpStream.eof())
        {
        ExpStream >> facevalue >> TestString >> Name;
      if (TestString.length()>1)
                {
                        NewNode = new ProbeNode;
           NewNode->DataPointer = new ProbeSynth;
                        transform(TestString);
              NewNode->DataPointer->Probe = TestString;
          NewNode->DataPointer->Name = Name;
                        NewNode->DataPointer->Location = facevalue;
                        NewNode->DataPointer->SynthesizeFastFluff(MutateValue);
                        this->DataSet.InsertAfterCurrent(NewNode);
                }
      counter++;
      if (counter%100==0)
        cout << counter << TAB << Name << TAB << TestString << endl;
        }
        ExpStream.close();
        cout << DataSet.Cost << endl;
}

TSPClass::LoadChipData(string FileName)
{
        // build the basic datalist
        ifstream ExpStream;
        ExpStream.open(FileName.c_str(), ios::in);

        if (ExpStream.bad())
        {
                cout << endl << "Experimental data file specified does not exist!";
                cout << endl << endl;
                exit(0);
        }
        long probeloop = 0;
        string TestString;
   ProbeNode *NewNode;
        long test; // soak up number

        while (!ExpStream.eof())
        {
                ExpStream >> TestString >> test;
      if (TestString.length()>1)
```

```
        {
                NewNode = new ProbeNode;
            NewNode->DataPointer = new ProbeSynth;
                        NewNode->DataPointer->Probe = TestString;
                        NewNode->DataPointer->SynthesizeMutant(test-1);
                        this->DataSet.InsertAfterCurrent(NewNode);
                  }
         }
         ExpStream.close();
         cout << DataSet.Cost << endl;
}

TSPClass::OutputTour(string FileName)
{
         ofstream OutStream;
         OutStream.open(FileName.c_str(), ios::out);

         this->BestTour.Output(OutStream);

         OutStream.close();
}

TSPClass::ListTour(string FileName)
{
         ofstream OutStream;
         OutStream.open(FileName.c_str(), ios::out);

         this->BestTour.List(OutStream);

         OutStream.close();
}

TSPClass::AppendTour(string FileName)
{
         ofstream OutStream;
         OutStream.open(FileName.c_str(), ios::app);

         this->BestTour.Output(OutStream);

         OutStream.close();
}

static void do_gematch(long Value)
{
         TSPClass Example;

         Example.LoadMisMatchFluffData("gematch.prb", 10-1);
         Example.GenerateTourByInsertionWithModifier(Value);
         Example.OutputTour("gematch.flf");
}

static void do_yematch()
{
         TSPClass Example;

         Example.LoadExpressionData("yematch.prb", 10-1);
         Example.GenerateTourByInsertionWithModifier(1);
         Example.OutputTour("yematch.tsp");
}

static void do_yematch_local(long UnitMatch)
{
         TSPClass Example;

         Example.LoadExpressionDataByUnit("yematch.prb", 10-1, UnitMatch);
         Example.GenerateTourByInsertionWithModifier(1);
         Example.AppendTour("yematch.lsp");
}

static void do_yematch_local_pool(long UnitMatch)
{
```

```
        TSPClass Example;

        Example.LoadExpressionDataByUnit("yematch.prb", 10-1, UnitMatch);
        Example.GenerateTourByClosestPool();
        Example.AppendTour("yematch.csp");
}

static void do_hummatch_local(long UnitMatch)
{
        TSPClass Example;

        Example.LoadExpressionData("ha.prb", 13-1);
        Example.GenerateQuickTourByInsertion(2048);
        Example.OutputTour("ha.tsp");

        //while (Example.GenerateTourByInsertionAndDeletion())
                //Example.AppendTour("ha.lsp");
}

static void do_fast_gematch(long Value)
{
        TSPClass Example;

        Example.LoadMisMatchFastFluffData("gematch.prb", 10-1);
        Example.GenerateTourByInsertionWithModifier(Value);
        Example.OutputTour("gefast.flf");
}

static void do_fast_pool_gematch(long Value)
{
        TSPClass Example;

        Example.LoadMisMatchFastFluffData("gematch.tny", 10-1);
        Example.GenerateTourByClosestPool();
        Example.OutputTour("gefast.pl");
}

static void do_mito(long Value)
{
        TSPClass Example;

        Example.LoadMutantData("mt9566.prb", 10-1);
        Example.GenerateTourByInsertionWithModifier(Value);
        //Example.ImproveTourByReplacement(1000);
        string Test;
        Test = "mt9566.";
   char ctest = 'a'+Value;
        Test += ctest;
   Example.OutputTour(Test);
}


static void do_hiv(long Value)
{
        TSPClass Example;

        Example.LoadChipData("hv430a.prb");
        Example.GenerateTourByInsertionWithModifier(Value);
        //Example.ImproveTourByReplacement(1000);
        string Test;
   Test = "hv430.";
        char ctest = 'a'+Value;
   Test += ctest;
        Example.OutputTour(Test);
}

static void do_new_ge()
{
        TSPClass Example;
```

```
        Example.LoadExpressionFluffData("i:\\data\\metrix\\ehubbe\\design\\nmisc\\gem01\\gem01.prb",
10-1);
        Example.GenerateTourByInsertionWithModifier(1);
        Example.OutputTour("gem01.flf");
}


static void do_noise()
{
        TSPClass Example;

        Example.LoadData("noisea8.20");
        Example.GenerateTourByInsertionWithModifier(1);
        Example.ListTour("na8_20.tsp");
}

static void do_noise_two()
{
        TSPClass Example;

        Example.LoadData("c:\\cover\\noisea8.16");
        Example.GenerateTourByInsertionWithModifier(1);
        Example.ListTour("na8_16.tsp");
}

static void do_noise_three()
{
        TSPClass Example;

        Example.LoadData("c:\\cover\\noisea8.18");
        Example.GenerateTourByInsertionWithModifier(1);
        Example.ListTour("na8_18.tsp");
}

static void do_cost_one()
{
        TSPClass Example;

        Example.LoadData("c:\\genius\\testa8.20");
        Example.GenerateTourByInsertionWithModifier(1);
        Example.ListTour("ca8_20.tsp");
}

static void do_cost_two()
{
        TSPClass Example;

        Example.LoadData("c:\\genius\\ca8_20.prb");
        Example.GenerateTourByInsertionWithModifier(1);
        Example.ListTour("cba8_20.tsp");
}

static void do_cost_quick()
{
        TSPClass Example;

        Example.LoadData("c:\\genius\\noise\\ca8_20.rnd");
        Example.GenerateQuickTourByInsertion(1024);
        Example.ListTour("cqa8_20.45");
}

static void do_rat_local(long UnitMatch)
{
        TSPClass Example;

        Example.LoadExpressionData("r:\\alldes\\cdesign\\ter09\\included_probes\\normal_probes.txt",
13 -1 );
        Example.LoadExpressionData("r:\\alldes\\cdesign\\ter09\\included_probes\\sense\\reverse_comp
_seqs.txt", 13 -1);
//      Example.LoadSingleExpressionData("r:\\alldes\\cdesign\\eo10191\\eoshu02.dat");
//      Example.GenerateQuickTourByInsertion(140000);
```

```
        Example.GenerateQuickTourByInsertion(20480);
        Example.OutputTour("r:\\alldes\\cdesign\\ter09\\full_fix.tsp");

        //while (Example.GenerateTourByInsertionAndDeletion())
                //Example.AppendTour("ha.lsp");
}

main()
{
        do_rat_local(0);
//      do_fast_pool_gematch(1);
//      do_yematch();
//      for (long i=1; i<104; i++)
//              do_yematch_local_pool(i);
//      do_gematch(1);
//    do_new_ge();
//      do_noise();
//      do_noise_two();
//      do_noise_three();
//      do_cost_quick();
}
```

Appendix B

Edgeopt.cpp

Attorney Docket 3296.1

Inventor: Earl A. Hubbell


```
// Edge Optimizer
#include <cstring.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <dir.h>
#include <time.h>
#include <math.h>

#define TAB '\t'
#define MXNAME 40
#define MXLINE 1000

#define TRUE 1
#define FALSE 0
#define MXSEQ 45
#define MXFEATURE 45
#define MXQUALIFIER 45

// Edge Optimizer Story
// 1) Strip off all Valid Blocks
// 2) Put Valid Blocks On to Minimize Edges

// Input: Cdl file, Ret file, Parameters
// Output: Twisted Cdl file and Ret file

char complement(char base)
{
        if (base=='A')
        return('T');
    if (base=='C')
        return('G');
    if (base=='G')
        return('C');
    if (base=='T')
        return('A');
    return(base);
}


class EntryClass{
// what CDL information is associated with everything
public:
        char sequence[MXSEQ];
        int destype;
        char feature[MXFEATURE];
```

```
        char qualifier[MXQUALIFIER];
        int expos;
        int     endpos;
        int pos;
        char pbase[MXFEATURE], tbase[MXFEATURE];
    int finishpos;
    int fixed;
    int variable;
        int unit, block;
    long atom;
    int repeat;
    int seqno;
    long layout;
        char locus[MXFEATURE];
    char accession[MXFEATURE];
    EntryClass(){Initialize();};
    Initialize();
    LineScan(char *);
    DumpLine(FILE *fp, int i, int j);
    DumpMut(FILE *fp);
};


EntryClass::Initialize()
{
        strcpy(sequence, "");
        destype = 0;
        strcpy(feature, "");
        strcpy(qualifier, "");
    expos = 0;
    pos = 0;
    strcpy(pbase,"!");
    strcpy(tbase,"!");
    unit = 0;
    block = 0;
    atom = 0;
}

EntryClass::LineScan(char *Line)
{
        int X,Y;
        static  char    PROBE[MXLINE];
        int     DESTYPE;
        static  char    FEATURE[MXLINE],
                        QUALIFIER[MXLINE];
        int     EXPOS;
        char    TBASE[MXLINE];
        int     ENDPOS,
                        POSITION;
        char  PBASE[MXLINE];
        int     FINISHPOS,
                        FIXED,
                        VARIABLE,
                        UNIT,
                        BLOCK,
                        REPEAT,
                        SEQNO,
                        LAYOUT;
    long ATOM;
        static  char  ACCESSION[MXLINE],
                        LOCUS[MXLINE];

                        sscanf(Line, "%d %d %s %d %s %s %d %s %d %d %s %d %d %d %d %d %ld %d %d %d %s
%s",
                                                                        &X,
                                                                        &Y,
                                                                        PROBE,
                                                                        &DESTYPE,
                                                                        FEATURE,
                                                                        QUALIFIER,
                                                                        &EXPOS,
```

```
                                                                TBASE,
                                                                &ENDPOS,
                                                                &POSITION,
                                                                PBASE,
                                                                &FINISHPOS,
                                                                &FIXED,
                                                                &VARIABLE,
                                                                &UNIT,
                                                                &BLOCK,
                                                                &ATOM,
                                                                &REPEAT,
                                                                &SEQNO,
                                                                &LAYOUT,
                                                                LOCUS,
                                                                ACCESSION);
                strcpy(sequence, PROBE);
                destype = DESTYPE;
                strcpy(feature, FEATURE);
                strcpy(qualifier, QUALIFIER);
                strcpy(locus, LOCUS);
                strcpy(accession, ACCESSION);
                expos = EXPOS;
                endpos = ENDPOS;
                finishpos = FINISHPOS;
        fixed = FIXED;
        variable = VARIABLE;
                repeat = REPEAT;
                seqno = SEQNO;
                layout = LAYOUT;

                strcpy(tbase, TBASE);
                strcpy(pbase, PBASE);
                unit = UNIT;
                block = BLOCK;
                atom = ATOM;
                pos = POSITION;
}

EntryClass::DumpLine(FILE *fp, int i, int j)
{
        fprintf(fp,
"%d\t%d\t%s\t%d\t%s\t%s\t%d\t%s\t%d\t%d\t%s\t%d\t%d\t%d\t%d\t%d\t%ld\t%d\t%d\t%d\t%s\t%s\n",
                                                                i,
                                                                j,
                                                                sequence,
                                                                destype,
                                                                feature,
                                                                qualifier,
                                                                expos,
                                                                tbase,
                                                                endpos,
                                                                pos,
                                                                pbase,
                                                                finishpos,
                                                                fixed,
                                                                variable,
                                                                unit,
                                                                block,
                                                                atom,
                                                                repeat,
                                                                seqno,
                                                                layout,
                                                                locus,
                                                                accession);
}

EntryClass::DumpMut(FILE *fp)
{
char tempc;

        if (destype==0)
```

```
    {
        fprintf(fp, "-");
        return(TRUE);
    }
    if (abs(destype)<100)
    {
        if (destype>0)
                fprintf(fp, "C");
        else
                fprintf(fp, "X");
        return(TRUE);
    }
        if (strlen(pbase)>1)
    {
        fprintf(fp, "I");
        return(TRUE);
    }
    if (pbase[0]=='!')
    {
        fprintf(fp, "D");
        return(TRUE);
    }
    if (destype>0)
    {
        tempc = complement(pbase[0]);
    }
    else
        tempc = pbase[0];
    if (tempc==tbase[0])
    {
                fprintf(fp, "%c", tempc);
        return(TRUE);
    }
    fprintf(fp, " ");
    return(TRUE);
}


class SynthClass{
// how things are built
public:
    char *synthesis;
    int synlength;
        SynthClass(){synthesis=NULL; synlength = 0;};
    Allocate(int);
    DeAllocate();
    Diff(SynthClass &);
    SetBit(char, int);
    char GetBit(int);
    GetLast();
    GetFirst();
    ~SynthClass();
};

SynthClass::~SynthClass()
{
        DeAllocate();
}

SynthClass::Allocate(int Size)
{
        synthesis = new char [Size]; // just a bitfield, really
    if (synthesis==NULL)
    {
        printf("Blow up! In SynthClass::Allocate");
        return(FALSE);
    }
    for (int i=0; i<Size; i++)
        synthesis[i] = 0; // Nothing, null, no bits set!
    synlength = Size;
        return(TRUE);
```

```
}

SynthClass::DeAllocate()
{
        if (synthesis!=NULL)
                delete[] synthesis;
    synlength = 0;
    synthesis = NULL;
}

SynthClass::SetBit(char value, int Which)
{
        if (synthesis!=NULL && Which<synlength && Which>-1)
                synthesis[Which] = value;
}

char
SynthClass::GetBit(int Which)
{
        if (synthesis!=NULL && Which<synlength && Which>-1)
                return(synthesis[Which]);
    else
        return(0);
}

SynthClass::Diff(SynthClass &Source)
{
        int count = 0;
    if (!(synthesis!=NULL && Source.synthesis!=NULL && synlength==Source.synlength))
        return(0);
        for (int i=0; i<synlength; i++)
    {
                count += (synthesis[i]!=Source.synthesis[i]);
    }
    return(count);
}

SynthClass::GetLast()
{
        for (int i=synlength-1; i>0; i--)
        if (synthesis[i]>0)
                    return(i);
}

SynthClass::GetFirst()
{
        for (int i=0; i<synlength; i++)
        if (synthesis[i]>0)
                    return(i);
}

class LocalDataClass{
public:
        int relx, rely;  // relative positions within a block
    int validflag; // newly added - can I move this for optimization?
        EntryClass cdldata;
    SynthClass retdata;
    LocalDataClass(){relx=rely=0;validflag = TRUE;};
    ~LocalDataClass();
    SetRelative(int, int);
    PrintLongSeq(FILE *fp);
};

LocalDataClass::~LocalDataClass()
{
        retdata.DeAllocate();
}

LocalDataClass::SetRelative(int i, int j)
{
        relx = i;
```

```
    rely = j;
}

LocalDataClass::PrintLongSeq(FILE *fp)
{
        int j, i=3;

        for (j=0; j<retdata.synlength; j++)
    {
        if (retdata.GetBit(j))
        {
                fprintf(fp, "%c", cdldata.sequence[i]);
            i++;
        }
        else
                fprintf(fp, ".");
    }
}



class BlockClass{
public:
        LocalDataClass **DataStack;
    int DataStackSize;
    int WSize, HSize; // rectangular grid
    BlockClass(){DataStack=NULL; DataStackSize = 0; WSize = HSize = 0;};
    Allocate(int);
    DeAllocate();
    ~BlockClass();
};

BlockClass::Allocate(int Size)
{
        DataStack = new LocalDataClass *[Size];
    if (DataStack==NULL)
        return(FALSE);
    for (int i=0; i<Size; i++)
        DataStack[i]=NULL;
    DataStackSize = Size;
}

BlockClass::DeAllocate()
{
        if (DataStack==NULL)
        return(TRUE);
    for (int i=0; i<DataStackSize; i++)
        if (DataStack[i]!=NULL)
        {
                printf("Error!  Data Leakage from Block!\n");
                        delete DataStack[i];
            DataStack[i] = NULL;
        }
    delete[] DataStack;
    DataStack = NULL;
    DataStackSize = 0;
    WSize = HSize = 0;
}

BlockClass::~BlockClass()
{
        DeAllocate();
}

class BlockStackClass{
public:
        BlockClass **BlockStack;
    long BlockCurSize;
    long BlockStackSize;
    BlockStackClass(){BlockStack=NULL; BlockStackSize = 0;BlockCurSize = 0;};
    Allocate(long);
```

```
    DeAllocate();
    ~BlockStackClass(){DeAllocate();};
    BlockClass *PutBlockOnStack(BlockClass *);
    BlockClass *RemoveBlock(long);
    BlockClass *TemporaryBlockFromStack(long);
    Swap(long, long);
    Shuffle();
};

BlockStackClass::Allocate(long Size)
{
        BlockClass ** TmpStack;

        TmpStack = new BlockClass * [Size];
    if (TmpStack==NULL)
        return(FALSE);
    long i;

    for (i=0; i<Size; i++)
        TmpStack[i]=NULL;
    for (i=0; i<BlockCurSize && i<BlockStackSize && BlockStack!=NULL; i++)
    {
        TmpStack[i] = BlockStack[i];
         BlockStack[i] = NULL;
    }
    if (BlockStack!=NULL)
        delete[] BlockStack;
    BlockStack = TmpStack;
    TmpStack = NULL;
    BlockStackSize = Size;
    return(TRUE);
}

BlockClass *
BlockStackClass::PutBlockOnStack(BlockClass *TempBlock)
{
        if (BlockCurSize<BlockStackSize)
    {
    }
    else
    {
        if (!Allocate(BlockStackSize+1000))
        {
                printf("Can't increase block stack\n");
                return(TempBlock); // upgrade stack
        }
    }
    BlockStack[BlockCurSize] = TempBlock;
    BlockCurSize++;
    return(NULL); // remove pointer
}

BlockStackClass::Swap(long Source, long Sink)
{
        BlockClass *TempBlock;

    TempBlock = BlockStack[Sink];
    BlockStack[Sink] = BlockStack[Source];
    BlockStack[Source] = TempBlock;
    TempBlock = NULL;
    return(TRUE);
}

BlockClass *
BlockStackClass::RemoveBlock(long WhichBlock)
{
        BlockClass *TempBlock;

    if (WhichBlock>=BlockCurSize || WhichBlock<0)
        return(NULL);
```

```
        TempBlock = BlockStack[WhichBlock];
    BlockCurSize--;
    BlockStack[WhichBlock] = BlockStack[BlockCurSize];
    BlockStack[BlockCurSize]=NULL;
    return(TempBlock);
}

BlockClass *
BlockStackClass::TemporaryBlockFromStack(long WhichBlock)
{
        BlockClass *TempBlock;

    if (WhichBlock>=BlockCurSize || WhichBlock<0)
        return(NULL);

        TempBlock = BlockStack[WhichBlock];
    return(TempBlock);
}

BlockStackClass::DeAllocate()
{
        if (BlockStack==NULL)
        return(TRUE);
    long i;
    for (i=0; i<BlockStackSize; i++)
    {
        if (BlockStack[i]!=NULL)
        {
                printf("Data Leakage from BlockStack\n");
            delete BlockStack[i];
            BlockStack[i] = NULL;
        }
    }
    delete[] BlockStack;
    BlockStack = NULL;
    BlockStackSize = 0;
    BlockCurSize = 0;
}

BlockStackClass::Shuffle()
{
        long t;
    long i;

        // randomly rearrange the stack to prevent bias
    for (i=BlockCurSize-1; i>0; i--)
    {
        t = rand()%32000;
        t = t*32000+(rand()%32000);
        t = t %(i+1);
                Swap(i,t);
    }
}

class ChipArrayClass{
public:
        BlockStackClass ValidBlockStack;
    LocalDataClass ***DataGrid;
    int Xdim;
    int Ydim;
    int SynthSteps;

    char retname[MXNAME];
    long NumOnes; // useful statistic

    int GlobalHeight;
    int MaxAllowed;
    int Radius;

    float LeakageHalfLife;
    int ScanRadius;
```

```
    int weightflag; // type of weights to use

    // constraints


    ChipArrayClass();
    ~ChipArrayClass();
    Allocate(int, int);
    DeAllocate();
    ReadCdl(char *, int);
    DumpCdl(char *);
    ReadRet(char *, int);
    DumpRet(char *);

    StripAreaToBlock(BlockClass *, int,int,int,int);
    CheckBlockFitToArea(BlockClass *, int, int);
        PutBlockInArea(BlockClass *, int, int);

    ValidMove(int, int);
    ValidLocation(int, int);
    Valid(int,int);
    ValidBlock(int,int,int,int);
    ValidTile(int,int,int,int);
    ValidBlank(int,int,int,int);

    CountDiff(LocalDataClass *,int,int);
    double CountEdges(BlockClass *, int, int);
    double CountWeightedEdges(BlockClass *, int, int);
    double CountEdgesFromStack(long, int, int);
    FindNextDiagonalSlot(int &, int &);
    FindNextHorizontalSlot(int &, int &);
    DiagonalReplacement(long);
    HorizontalReplacement(long);
    StripAllValidBlocks(int);
    PlaceBlockFromStack(long, int,int);
    SearchLocationWithStats(int,int,long, long &, double &, double &, double &);

    CountUnitInArea(long, int, int,int,int);
    ProximityCheckBlock(BlockClass *, int,int,int,int);
    ProximityCheckFromStack(long, int,int,int,int);

    StripBadProximityValues(int);
    PickRandomValidBlock(int &, int &, int, int);

    ReadInstructionFile(char *);
    InterpretInstructionLine(char *);
    GenerateMutFile(char *);
    SetUnits(long, long, int);
    SetArea(int, int, int, int, int);
    SetAntiArea(int,int,int,int,int);
    SetDestype(int, int);
    Shuffle();
    StripValidBlock(int,int,int);
    StripRandomBlocks(long, int);
    DoubleReplacement(long);
    GenerateDiffFile(char *);

    FindNextAggregateSlot(int &, int &);
    AggregateReplacement(long);
};

ChipArrayClass::Shuffle()
{
        ValidBlockStack.Shuffle();
}

ChipArrayClass::SetArea(int X, int Y, int tX, int tY, int value)
{
        int i,j;
        for (i=X; i<=tX; i++)
    {
```

```
        for (j=Y; j<=tY; j++)
        {
                if (Valid(i,j))
                        DataGrid[i][j]->validflag = value;
        }
    }
}

ChipArrayClass::SetAntiArea(int X, int Y, int tX, int tY, int value)
{
        // used to set all >but< a given physical area
    int i,j;
    for (i=0; i<Xdim; i++)
    {
        for (j=0; j<Ydim; j++)
        {
                if (Valid(i,j))
            {
                if (!(i>=X && i<=tX && j>=Y && j<=tY))
                        DataGrid[i][j]->validflag = value;
            }
        }
    }
}

ChipArrayClass::SetDestype(int destype, int value)
{
        int i,j;

        for (i=0; i<Xdim; i++)
    {
        for (j=0; j<Ydim; j++)
        {
                if (Valid(i,j))
            {
                if (destype==DataGrid[i][j]->cdldata.destype)
                {
                        DataGrid[i][j]->validflag = value; // can't move this one
                }
            }
        }
    }

}

ChipArrayClass::ValidMove(int X, int Y)
{
        if (DataGrid[X][Y]->validflag)
        return(TRUE);
    return(FALSE);
}

ChipArrayClass::ValidLocation(int X, int Y)
{
        if (X<0 || Y<0 || X>=Xdim || Y>=Ydim)
        return(FALSE);
        // nothing here yet
    return(TRUE);
}

ChipArrayClass::Valid(int X, int Y)
{
    if (!ValidLocation(X,Y))
        return(FALSE); // not a location we're allowed to move
    if (DataGrid[X][Y]==NULL)
        return(FALSE); // doesn't even exist!
    return(TRUE);
}

ChipArrayClass::SetUnits(long Start, long Finish, int value)
{
```

```
        int i,j;

        for (i=0; i<Xdim; i++)
    {
        for (j=0; j<Ydim; j++)
        {
              if (Valid(i,j))
            {
               if (Start<=DataGrid[i][j]->cdldata.unit && Finish>=DataGrid[i][j]->cdldata.unit)
                {
                      DataGrid[i][j]->validflag = value; // can't move this one
                }
            }
        }
    }
}


ChipArrayClass::ValidTile(int X, int Y, int Width, int Height)
{
        long U, A;
    int tx, ty,i,j;
    int xl, yl;
    xl = X+Width;
    yl = Y+Height;

        U = DataGrid[X][Y]->cdldata.unit;
    A = DataGrid[X][Y]->cdldata.atom;
//    if (Width==Height && Height==1)  // if we're moving control probes around!
//            return(TRUE);
        for (i=-1; i<Width+1; i++)
    {
        for (j=-1; j<Height+1; j++)
        {
                      tx = i+X;
            ty = j+Y;
            if (tx>=X && ty>=Y && tx<xl && ty<yl)
            {
               if (!Valid(tx,ty) || !ValidMove(tx,ty))   // if not valid we're unhappy
                      return(FALSE);
                if (U!=DataGrid[tx][ty]->cdldata.unit || A!=DataGrid[tx][ty]->cdldata.atom) // if
not same, we're unhappy
                      return(FALSE);
            }
            else
            {
               if (Valid(tx,ty))
                {
                      if (U==DataGrid[tx][ty]->cdldata.unit && A==DataGrid[tx][ty]->cdldata.atom)
// if outside same block unhappy
                      return(FALSE);
                }
            }
        }
    }
    return(TRUE);
}

ChipArrayClass::ValidBlank(int X, int Y, int Width, int Height)
{
        int i,j, tx,ty;

        // valid set of blanks
    for (i=0; i<Width; i++)
        for (j=0; j<Height; j++)
        {
              tx = i+X;
            ty = j+Y;
            if (!Valid(tx,ty) || !ValidMove(tx,ty))
               return(FALSE);
            if (DataGrid[tx][ty]->cdldata.destype!=0)
```

```
                return(FALSE);
            }
        return(TRUE);
    }

    ChipArrayClass::ValidBlock(int X, int Y, int Width, int Height)
    {

            if (!Valid(X,Y))
            return(FALSE);
        if (!ValidMove(X,Y))
            return(FALSE);
        if (ValidTile(X,Y,Width, Height))
            return(TRUE);
        if (ValidBlank(X,Y, Width, Height)) // allow blank blocks to be moved, if validflag set for
    destype 0
            return(TRUE);

        return(FALSE);
    }

    ChipArrayClass::PickRandomValidBlock(int &X, int &Y, int Width, int Height)
    {
            long counter = 0;
        long Limit = 10000;
            X = rand()%Xdim;
        Y = rand()%Ydim;
            while(!ValidBlock(X,Y, Width, Height) && counter<Limit)
        {
            X = rand()%Xdim;
             Y = rand()%Ydim;
             counter++;
        }
        if (counter==Limit)
            return(FALSE); // can't find one in reasonable time!
        return(TRUE);
    }


    ChipArrayClass::CountDiff(LocalDataClass *TestData, int X, int Y)
    {
            if (!Valid(X,Y))
            return(0);  // doesn't exist or is off chip, so no problems!
        return(TestData->retdata.Diff(DataGrid[X][Y]->retdata));
    }

    double
    ChipArrayClass::CountEdges(BlockClass *TempBlock, int X, int Y)
    {
            int i,tx,ty;
        int count = 0;
            // count the edges, if this block is in this location
        // note that "interior" edges of blocks are >not< counted
            for (i=0; i<TempBlock->DataStackSize; i++)
        {
            if (TempBlock->DataStack[i]!=NULL)
             {
                    tx = X+TempBlock->DataStack[i]->relx;
                     ty = Y+TempBlock->DataStack[i]->rely;
                count +=CountDiff(TempBlock->DataStack[i], tx+1, ty);
                count +=CountDiff(TempBlock->DataStack[i], tx-1, ty);
                count +=CountDiff(TempBlock->DataStack[i], tx, ty-1);
                count +=CountDiff(TempBlock->DataStack[i], tx, ty+1);
             }
        }
        return(count);
    }

    double
    ChipArrayClass::CountWeightedEdges(BlockClass *TempBlock, int X, int Y)
    {
```

```
        int i,tx,ty;
    int rangex,rangey;
    double count = 0;
    double lcount;
    double distance;
        // count the edges, if this block is in this location
    // note that "interior" edges of blocks are >not< counted
        for (i=0; i<TempBlock->DataStackSize; i++)
    {
        if (TempBlock->DataStack[i]!=NULL)
        {
                for (rangex = -1*ScanRadius; rangex<=ScanRadius; rangex++)
            {
               for (rangey=-1*ScanRadius; rangey<=ScanRadius; rangey++)
                {
                      if (rangex!=0 || rangey!=0)
                   {
                            distance = (rangex*rangex)+(rangey*rangey);
                         distance = sqrt(distance);
                                  tx = X+TempBlock->DataStack[i]->relx+rangex;
                               ty = Y+TempBlock->DataStack[i]->rely+rangey;
                                lcount =CountDiff(TempBlock->DataStack[i], tx, ty);
                         lcount*=pow(.5,LeakageHalfLife*distance);
                         count +=lcount;
                      }
                }
            }
        }
    }
    return(count);
}


ChipArrayClass::CountUnitInArea(long Unit, int X, int Y, int Width, int Height)
{
        int i,j;
    int tx, ty;
    int count =0;
    long tatom=-100; // unlikely in any real unit

        for (i=0; i<Width; i++)
    {
        tatom = -100; // start over with each vertical stripe - assumes vertical "atoms"
        for (j=0; j<Height; j++)
        {
                tx = X+i;
            ty= Y+j;
            if (Valid(tx,ty))
            {
               if (DataGrid[tx][ty]->cdldata.unit==Unit)
                {
                       // works because we're scanning vertically
                       if (tatom!=DataGrid[tx][ty]->cdldata.atom) // only count a given unit/atom
once
                                count++;
                    tatom = DataGrid[tx][ty]->cdldata.atom;
                }
            }
        }
    }
    return(count);
}

ChipArrayClass::ProximityCheckBlock(BlockClass *TempBlock, int X, int Y, int Width, int Height)
{
        return(CountUnitInArea(TempBlock->DataStack[0]->cdldata.unit,X,Y,Width,Height));
}

ChipArrayClass::ProximityCheckFromStack(long Which, int X, int Y, int Width, int Height)
{
        BlockClass *TempBlock;
```

```
    int count;

    TempBlock = ValidBlockStack.TemporaryBlockFromStack(Which);
    if (TempBlock==NULL)
        return(16000); // big error
    count = ProximityCheckBlock(TempBlock, X, Y,Width, Height);
    TempBlock = NULL;
    return(count);
}

double
ChipArrayClass::CountEdgesFromStack(long Which, int X, int Y)
{
        BlockClass *TempBlock;
    double count;

    TempBlock = ValidBlockStack.TemporaryBlockFromStack(Which);
    if (TempBlock==NULL)
        return(16000); // big error
    if (!weightflag)
            count = CountEdges(TempBlock, X, Y);
    else
        count = CountWeightedEdges(TempBlock, X,Y);
    TempBlock = NULL;
    return(count);
}

ChipArrayClass::FindNextDiagonalSlot(int &X, int &Y)
{
        // look for a NULL entry
    while (Y<Ydim && DataGrid[X][Y]!=NULL)
    {
        X-=1;
        Y+=1;
        if (X<0 || Y>=Ydim)
        {
               X = Y+X+1;
            Y = 0;
        }
        if (X>=Xdim)
        {
               Y = X-(Xdim-1);
            X = Xdim-1;
        }
    }
    if (Y==Ydim) // ran out of room!
        return(FALSE);
    else
        return(TRUE);
}

ChipArrayClass::FindNextHorizontalSlot(int &X, int &Y)
{
        // look for a NULL entry
    while (Y<Ydim && DataGrid[X][Y]!=NULL)
    {
        X +=1;
        if (X>=Xdim)
        {
               X = 0;
            Y++;
        }
    }
    if (Y==Ydim) // ran out of room!
        return(FALSE);
    else
        return(TRUE);
}

ChipArrayClass::FindNextAggregateSlot(int &X, int &Y)
{
```

```
        int total;
        total = X;
    if (Y>total)
        total = Y;
    while (Y<Ydim && DataGrid[X][Y]!=NULL)
    {
        // look for slots
        if (X>Y)
               Y++; // move vertically
        else
                if (X<=Y)
                       X--;  // basic zero moves
        if (X<0)
        {
               X=Y+1; // add one to total
            Y=0;
        }
        if (X>=Xdim)
        {
               Y = X;
            X=Xdim-1;
        }
    }
    if (Y>=Ydim)
        return(FALSE);
    else
            return(TRUE);
}

ChipArrayClass::PlaceBlockFromStack(long Which, int X, int Y)
{
       BlockClass *TempBlock;

              TempBlock = ValidBlockStack.RemoveBlock(Which);
            if (TempBlock!=NULL)
            {
                            if (PutBlockInArea(TempBlock, X,Y))
                     TempBlock = NULL; // keep wacky pointers from drifting
                 else
                 {
                       printf("Failure to fit block in area: %d %d\n", X, Y);
                    TempBlock = ValidBlockStack.PutBlockOnStack(TempBlock); // throw back on stack
                 }
            }
            else
               printf("Failure to get from stack! %d %d\n", X,Y);
}

ChipArrayClass::SearchLocationWithStats(int X, int Y, long searchlimit, long &Best, double &bestc,
double &avg, double &worstc)
{
       long search;
    long count = 0;
    double c;

        Best = ValidBlockStack.BlockCurSize-1; // which one
        bestc = CountEdgesFromStack(ValidBlockStack.BlockCurSize-1,X,Y);
               avg = bestc;
        worstc = bestc;
        count = 1;
        for (search=1; search<searchlimit && search<ValidBlockStack.BlockCurSize; search++)
        {
                        c = CountEdgesFromStack(ValidBlockStack.BlockCurSize-1-search,X,Y); // what
if we put here?
            avg +=c;
            if (c<bestc)
            {
               bestc = c;
                Best = ValidBlockStack.BlockCurSize-1-search;
            }
            if (c>worstc)
```

```
            {
               worstc=c;
            }
            count++;
         }
         avg /=count; // number actually searched
        return(TRUE);
}


ChipArrayClass::StripBadProximityValues(int H)
{
        int i,j;
    long U;
    int c;
    BlockClass *TempBlock;

    GlobalHeight = H;

        for (i=0; i<Xdim; i++)
    {
                for (j=0; j<Ydim; j++)
        {
                if (ValidBlock(i,j,1, H))  // note that blanks could mess this up badly!
            {
                U = DataGrid[i][j]->cdldata.unit;
                        c = CountUnitInArea(U,i-Radius, j-Radius, 2*Radius+1, 2*Radius+1);
                 if (c>MaxAllowed)
                        StripValidBlock(i,j,H);
            }
        }
    }
    // now we've got all our trouble removed from the chip
    printf("Bad Proximity values: %d %d %d %ld\n", H, Radius, MaxAllowed,
ValidBlockStack.BlockCurSize);
    return(TRUE);
}



ChipArrayClass::DoubleReplacement(long searchlimit)
{
        // idea is to "dilute" any bad values
    // this only works if the chip is sufficiently large
    // and there are sufficiently few bad items
        StripBadProximityValues(GlobalHeight);
        while (ValidBlockStack.BlockCurSize>0)
    {
         StripRandomBlocks(ValidBlockStack.BlockCurSize+100, GlobalHeight); // get some good random
locations freed up
         Shuffle(); // rearrange life
         DiagonalReplacement(searchlimit); // put 'em back, - if too much search, goes back exactly
to bad spots
                StripBadProximityValues(GlobalHeight); // find out if we've got them all
    }
}



ChipArrayClass::DiagonalReplacement(long searchlimit)
{
        // replaces blocks from stack onto the chip
    int X, Y;
    long Best;
    double EdgesAdded = 0.1;
    double TotalAdded = 0.1;
    double AvgEdges =0.1;
    double WorstEdges=0.1;
    double bestc;
    double avg;
```

```
    double worstc;
    long Report = 1000000;

    Report /=searchlimit;
    if (Report>1000)
       Report=1000;

    X=Y=0;
    while (FindNextDiagonalSlot(X,Y))
    {
       // found a location where a block was removed
        SearchLocationWithStats(X,Y,searchlimit, Best, bestc, avg, worstc);
        // found the best thing to put there

        // and so put it there!
        PlaceBlockFromStack(Best,X,Y);

        EdgesAdded += bestc; AvgEdges += avg; WorstEdges += worstc; TotalAdded ++;

        if (ValidBlockStack.BlockCurSize%Report==0)
              printf("At: %d %d %ld %lf %lf %lf\r", X, Y, ValidBlockStack.BlockCurSize,
EdgesAdded/(2*TotalAdded), EdgesAdded/AvgEdges,EdgesAdded/WorstEdges);
    }
    printf("\nAt: %d %d %ld %lf %lf %lf\n", X, Y, ValidBlockStack.BlockCurSize,
EdgesAdded/(2*TotalAdded), EdgesAdded/AvgEdges,EdgesAdded/WorstEdges);
    return(TRUE);
}

ChipArrayClass::AggregateReplacement(long searchlimit)
{
       // replaces blocks from stack onto the chip
    int X, Y;
    long Best;
    double EdgesAdded = 0.1;
    double TotalAdded = 0.1;
    double AvgEdges =0.1;
    double WorstEdges=0.1;
    double bestc;
    double avg;
    double worstc;
    long Report = 1000000;

    Report /=searchlimit;
    if (Report>1000)
       Report=1000;

    X=Y=0;
    while (FindNextAggregateSlot(X,Y) && (ValidBlockStack.BlockCurSize>0))
    {
       // found a location where a block was removed
        SearchLocationWithStats(X,Y,searchlimit, Best, bestc, avg, worstc);
        // found the best thing to put there

        // and so put it there!
        PlaceBlockFromStack(Best,X,Y);

        EdgesAdded += bestc; AvgEdges += avg; WorstEdges += worstc; TotalAdded ++;

        if (ValidBlockStack.BlockCurSize%Report==0)
              printf("At: %d %d %ld %lf %lf %lf\r", X, Y, ValidBlockStack.BlockCurSize,
EdgesAdded/(2*TotalAdded), EdgesAdded/AvgEdges,EdgesAdded/WorstEdges);
    }
    printf("\nAt: %d %d %ld %lf %lf %lf\n", X, Y, ValidBlockStack.BlockCurSize,
EdgesAdded/(2*TotalAdded), EdgesAdded/AvgEdges,EdgesAdded/WorstEdges);
    return(TRUE);
}


ChipArrayClass::HorizontalReplacement(long searchlimit)
{
```

```
        // replaces blocks from stack onto the chip
    int X, Y;
    long Best;
    double EdgesAdded = 0.1;
    double TotalAdded = 0.1;
    double AvgEdges =0.1;
    double WorstEdges=0.1;
    double bestc;
    double avg;
    double worstc;
    long Report = 1000000;

    Report /=searchlimit;
    if (Report>1000)
        Report=1000;

    X=Y=0;
    while (FindNextHorizontalSlot(X,Y))
    {
        // found a location where a block was removed
        SearchLocationWithStats(X,Y,searchlimit, Best, bestc, avg, worstc);
        // found the best thing to put there

        // and so put it there!
        PlaceBlockFromStack(Best,X,Y);

        EdgesAdded += bestc; AvgEdges += avg; WorstEdges += worstc; TotalAdded++;

        if (ValidBlockStack.BlockCurSize%Report==0)
              printf("At: %d %d %ld %lf %lf %lf\r", X, Y, ValidBlockStack.BlockCurSize,
EdgesAdded/(2*TotalAdded), EdgesAdded/AvgEdges,EdgesAdded/WorstEdges);
    }
    printf("\nAt: %d %d %ld %lf %lf %lf\n", X, Y, ValidBlockStack.BlockCurSize,
EdgesAdded/(2*TotalAdded), EdgesAdded/AvgEdges,EdgesAdded/WorstEdges);
    return(TRUE);
}

ChipArrayClass::StripValidBlock(int X, int Y, int H)
{
       BlockClass *TempBlock;

    if (ValidBlock(X,Y, 1, H))
    {
        TempBlock = new BlockClass;
        StripAreaToBlock(TempBlock, X,Y,1,H); // take probe from chip
        TempBlock = ValidBlockStack.PutBlockOnStack(TempBlock);
        if (TempBlock!=NULL)
        {
              printf("Failure to put on stack! %d %d\n", X,Y);
        }
    }
}

ChipArrayClass::StripAllValidBlocks(int H)
{
       int i,j;
    BlockClass *TempBlock;

       for (i=0; i<Xdim; i++)
    {
        for (j=0; j<Ydim; j++)
        {
              StripValidBlock(i,j,H);
        }
        printf("Stripped: %ld\r", ValidBlockStack.BlockCurSize);
    }
}

ChipArrayClass::StripRandomBlocks(long Num, int H)
{
       long i;
```

```
    int X,Y;

        for (i=0; i<Num; i++)
    {
        if(PickRandomValidBlock(X,Y,1,H))
                StripValidBlock(X,Y,H);
    }
    return(TRUE);
}


ChipArrayClass::StripAreaToBlock(BlockClass *TempBlock, int X, int Y, int Width, int Height)
{
        if (X+Width>Xdim || Y+Height>Ydim || X<0 || Y<0)
        return(FALSE);
        // strip an area of the chip into a block
    TempBlock->Allocate(Width*Height);
    TempBlock->WSize = Width;
    TempBlock->HSize = Height;

    int counter = 0;

    int i,j;
    for (i=0; i<Width; i++)
    for (j=0; j<Height; j++)
    {
        TempBlock->DataStack[counter] = DataGrid[X+i][Y+j];
        DataGrid[X+i][Y+j] = NULL; // removed
        TempBlock->DataStack[counter]->SetRelative(i,j);
        counter++;
    }
}

ChipArrayClass::CheckBlockFitToArea(BlockClass *TempBlock, int X, int Y)
{
        int valid = TRUE;
    int i;
    int tx, ty;

    if (TempBlock==NULL)
        return(FALSE);

        for (i=0; i<TempBlock->DataStackSize && valid; i++)
    {
        if (TempBlock->DataStack[i]!=NULL)
        {
                tx = X+TempBlock->DataStack[i]->relx;
                ty = Y+TempBlock->DataStack[i]->rely;
            if (tx<Xdim && ty<Ydim && tx>-1 && ty>-1)
                            if (DataGrid[tx][ty]!=NULL)
                    valid = FALSE;
                else
                    valid = TRUE;
            else
                valid = FALSE;
        }
    }
    return(valid);
}

ChipArrayClass::PutBlockInArea(BlockClass *TempBlock, int X, int Y)
{
        int i;
    int tx, ty;
        if (!CheckBlockFitToArea(TempBlock, X, Y))
        return(FALSE);
        for (i=0; i<TempBlock->DataStackSize; i++)
    {
        if (TempBlock->DataStack[i]!=NULL)
        {
                tx = X+TempBlock->DataStack[i]->relx;
```

```
                ty = Y+TempBlock->DataStack[i]->rely;
                        DataGrid[tx][ty] = TempBlock->DataStack[i];
            TempBlock->DataStack[i]=NULL;
        }
    }
        TempBlock->DeAllocate(); // toast!
        return(TRUE);
}

ChipArrayClass::ChipArrayClass()
{
        DataGrid = NULL;
    Xdim = Ydim = SynthSteps = 0;
    Radius = 9;
        MaxAllowed = 4;
    GlobalHeight = 2;

    ScanRadius = 1;
    LeakageHalfLife = 1;
    weightflag = 0;
}

ChipArrayClass::~ChipArrayClass()
{
        DeAllocate();
}

ChipArrayClass::Allocate(int X, int Y)
{
        DataGrid = new LocalDataClass ** [X];
    if (DataGrid==NULL)
        return(FALSE);
    int i,j;

    for (i=0; i<X; i++)
    {
        DataGrid[i] = new LocalDataClass * [Y];
        if (DataGrid[i]==NULL)
                return(FALSE);
        for (j=0; j<Y; j++)
        {
                DataGrid[i][j] = new LocalDataClass; // featherweight objects
            if (DataGrid[i][j]==NULL)
                return(FALSE);
        }
    }
    Xdim = X;
    Ydim = Y;
    return(TRUE);
}

ChipArrayClass::DeAllocate()
{
        if (DataGrid==NULL)
        return(TRUE);
        int i,j;
        for (i=0; i<Xdim; i++)
    {
        for (j=0; j<Ydim && DataGrid[i]!=NULL; j++)
        {
                if (DataGrid[i][j]!=NULL)
                        delete DataGrid[i][j];
        }
        if (DataGrid[i]!=NULL)
                delete[] DataGrid[i];
    }
    delete[] DataGrid;
    DataGrid = NULL;
    Xdim = Ydim = 0;
    SynthSteps = 0;
}
```

```
ChipArrayClass::ReadCdl(char *FileName, int realflag)
{
        FILE *ifp;
    int maxX, maxY, X, Y;
    char datastring[MXLINE];
    int flag=TRUE;

    if (realflag)
         flag = ReadCdl(FileName, FALSE);
    if (!flag)
        return(FALSE);
        ifp = fopen(FileName, "rt");
    if (NULL==ifp)
    {
        printf("Unable to open: %s\n", FileName);
         exit(1);
    }
    fgets(datastring, MXLINE, ifp);
    maxX = 0;
    maxY = 0;
    while (!feof(ifp) && !ferror(ifp))
    {
        fgets(datastring, MXLINE, ifp);
         if (feof(ifp) || ferror(ifp))
               break;
         sscanf(datastring, "%d %d", &X, &Y);
         if (X>maxX)
               maxX=X;
         if (Y>maxY)
               maxY=Y;
         if (realflag)
                DataGrid[X][Y]->cdldata.LineScan(datastring);
         if (X==0)
                printf("%d\r", Y);
    }
    fclose(ifp);
    if(!realflag)
    {
           maxX++;
           maxY++;
        flag = Allocate(maxX, maxY);
         return(flag);
    }
    return(TRUE);
}

ChipArrayClass::DumpCdl(char *FileName)
{
        FILE *fp;
    int i,j;

        fp = fopen(FileName, "wt");
    fprintf(fp,
"X\tY\tPROBE\tDESTYPE\tFEATURE\tQUALIFIER\tEXPOS\tTBASE\tENDPOS\tPOSITION\tPBASE\tFINISHPOS\tFIXED\t
VARIABLE\tUNIT\tBLOCK\tATOM\tREPEAT\tSEQNO\tLAYOUT\tACCESSION\tLOCUS\n");
        for (j=0; j<Ydim; j++)
    {
               for (i=0; i<Xdim; i++)
         {
               if (DataGrid[i][j]!=NULL)
                      DataGrid[i][j]->cdldata.DumpLine(fp, i,j);
           else
           {
              printf("Null value in grid %d %d\n", i,j);
           }
         }
         printf("OutCdl: %d\r", j);
    }
        fclose(fp);
```

```
}

ChipArrayClass::GenerateMutFile(char *FileName)
{
        FILE *fp;
    int i,j;

        fp = fopen(FileName, "wt");
        for (j=0; j<Ydim; j++)
    {
                for (i=0; i<Xdim; i++)
        {
                if (DataGrid[i][j]!=NULL)
            {
                                DataGrid[i][j]->cdldata.DumpMut(fp); // single descriptive character
            }
            else
            {
                fprintf(fp, "-");
                printf("Null value in grid %d %d\n", i,j);
            }
        }
        fprintf(fp, "\n");
        printf("MUT: %d\r", j);
    }
        fclose(fp);
}

ChipArrayClass::GenerateDiffFile(char *FileName)
{
        FILE *fp;
    int i,j;
    int tn,te,ts,tw;
    double n,e,s,w;
    double count;

    fp = fopen(FileName, "wt");
    for (j=0; j<Ydim; j++)
    {
        for (i=0; i<Xdim; i++)
        {
                if (Valid(i,j))
            {
                fprintf(fp, "X:%d\tY:%d\t", i,j);
                 tn=ts=tw=te =0;
                 if (Valid(i,j-1))
                        tn = DataGrid[i][j-1]->retdata.Diff(DataGrid[i][j]->retdata);
                 if (Valid(i,j+1))
                        ts = DataGrid[i][j+1]->retdata.Diff(DataGrid[i][j]->retdata);
                 if (Valid(i-1,j))
                        tw = DataGrid[i-1][j]->retdata.Diff(DataGrid[i][j]->retdata);
                 if (Valid(i+1,j))
                        te = DataGrid[i+1][j]->retdata.Diff(DataGrid[i][j]->retdata);
                 fprintf(fp, "N:%d\tE:%d\tS:%d\tW:%d\tT:%d\t",tn,te,ts,tw,tn+te+ts+tw);
                                fprintf(fp, "LAST:%d\t", DataGrid[i][j]->retdata.GetLast());
                 fprintf(fp, "BREADTH:%d\t", DataGrid[i][j]->retdata.GetLast()-DataGrid[i][j]-
>retdata.GetFirst());
                 DataGrid[i][j]->PrintLongSeq(fp);
                 fprintf(fp, "\t%s\n", DataGrid[i][j]->cdldata.qualifier);

                 count++;
                 n+=tn;
                 s+=ts;
                 e+=te;
                 w+=tw;
            }
            else
            {
                printf("Null value in grid %d %d\n", i,j);
            }
        }
```

```
        printf("Diff: %d %lf %lf %lf %lf %lf\r", j, n/count, e/count,s/count, w/count,
(n+e+s+w)/(4*count));
    }
    //fprintf(fp,"Diff: %d %lf %lf %lf %lf %lf\n", j, n/count, e/count,s/count, w/count,
(n+e+s+w)/(4*count));
    fclose(fp);
}

ChipArrayClass::ReadRet(char *FileName, int realflag)
{
        FILE *fp;
    int i,j, k;
    char dataline[MXLINE];
    long total;

    if (realflag)
       ReadRet(FileName, 0);
    fp = fopen(FileName, "rt");
    if (fp==NULL)
    {
       printf("Unable to open: %s\n", FileName);
        exit(1);
    }

    if (realflag)
    {
       for (i=0; i<Xdim; i++)
       for (j=0; j<Ydim; j++)
        {
              if (DataGrid[i][j]!=NULL)
                            DataGrid[i][j]->retdata.Allocate(SynthSteps);  // allocate this data
           else
              printf("Death by lack of allocation\n");
        }
    }


    k=-1;
    j=Ydim;
    total = 0;

    while (fgets(dataline, MXLINE, fp))
    {
       if (dataline[0]=='r')
        {
              sscanf(dataline, "reticle: %s", &retname);  // set up reticle template name
           retname[strlen(retname)-2] = '\0';
                     // initialize for reading next lines
           k++;
           printf("Reticle: %d\r", k);
           j=Ydim;
           continue;
        }
        if (strlen(dataline)<10 || dataline[0]==';')
              continue;
        if (dataline[0]=='0' || dataline[0]=='1')
        {
              j--;
              for (i=0; i<Xdim && j>-1; i++)
           {
              if (dataline[i]=='1')
               {
                     if (realflag)
                 {
                           DataGrid[i][j]->retdata.SetBit(1,k);
                 }
                 total ++;
               }
           }
        }
    }
```

```
    NumOnes = total;
    SynthSteps = k+1;

    fclose(fp);
}

ChipArrayClass::DumpRet(char *FileName)
{
        FILE *fp;
    int i,j,k;
    char dataline[MXLINE];

        fp = fopen(FileName, "wt");

    fprintf(fp, "; This file has been annealed to minimize edges\n");

    for (k=0; k<SynthSteps; k++)
    {
              fprintf(fp,"\n\n;base: X");
              fprintf(fp,"\nreticle: %s%02d", retname, (k+1));
              fprintf(fp,"\nR I 1 1 0 0 %d %d %d",Xdim, Ydim, 1);
       for (j=Ydim-1; j>-1; j--)
       {
           fprintf(fp, "\n");
              for (i=0; i<Xdim; i++)
           {
              if (DataGrid[i][j]==NULL)
               {
                     printf("Data leakage: %d %d\n", i,j);
               }
               else
                     if (DataGrid[i][j]->retdata.GetBit(k))
                            dataline[i] = '1';
                      else
                            dataline[i] = '0';
           }
           dataline[Xdim] = '\0';
           fprintf(fp, "%s", dataline);
       }
       fprintf(fp,";\n0;\n");
       printf("DUMPRET: %d\r", k);
    }
    fclose(fp);
}

ChipArrayClass::InterpretInstructionLine(char *Line)
{
        char TempStr[MXLINE];
    int height;
    long searchlimit;
    long start, finish;
    int tx,ty,x,y;
    int value;
    int destype;
    int radius, max;
    double dval;

       // read an instruction and do the appropriate thing
    if (Line[0]==';')
       return(TRUE); // comment
    sscanf(Line, "%s", TempStr); // pick off the initial piece
    if (!strcmp(TempStr, "READCDL:"))
    {
              sscanf(Line, "READCDL: %s", TempStr);
       ReadCdl(TempStr, TRUE);
       return(TRUE);
    }
    if (!strcmp(TempStr, "READRET:"))
    {
       sscanf(Line, "READRET: %s", TempStr);
       ReadRet(TempStr,1);
```

```
        return(TRUE);
    }
    if (!strcmp(TempStr, "DUMPCDL:"))
    {
        sscanf(Line, "DUMPCDL: %s", TempStr);
        DumpCdl(TempStr);
        return(TRUE);
    }
    if (!strcmp(TempStr, "DUMPRET:"))
    {
        sscanf(Line, "DUMPRET: %s", TempStr);
        DumpRet(TempStr);
        return(TRUE);
    }
    if (!strcmp(TempStr, "DUMPMUT:"))
    {
        sscanf(Line, "DUMPMUT: %s", TempStr);
        GenerateMutFile(TempStr);
        return(TRUE);
    }
    if (!strcmp(TempStr, "DUMPDIFF:"))
    {
        sscanf(Line, "DUMPDIFF: %s", TempStr);
        GenerateDiffFile(TempStr);
        return(TRUE);
    }
    if (!strcmp(TempStr, "STRIPBLOCKS:"))
    {
        sscanf(Line, "STRIPBLOCKS: %d", &height);
        GlobalHeight = height;
        StripAllValidBlocks(height);
        Shuffle();
        return(TRUE);
    }
    if (!strcmp(TempStr, "DIAGONALREPLACEMENT:"))
    {
        sscanf(Line, "DIAGONALREPLACEMENT: %ld", &searchlimit);
        DiagonalReplacement(searchlimit); // put all blocks back on chip
        return(TRUE);
    }
    if (!strcmp(TempStr, "HORIZONTALREPLACEMENT:"))
    {
        sscanf(Line, "HORIZONTALREPLACEMENT: %ld", &searchlimit);
        DiagonalReplacement(searchlimit); // put all blocks back on chip
        return(TRUE);
    }
    if (!strcmp(TempStr, "AGGREPLACEMENT:"))
    {
        sscanf(Line, "AGGREPLACEMENT: %ld", &searchlimit);
        AggregateReplacement(searchlimit); // put all blocks back on chip
        return(TRUE);
    }
    if (!strcmp(TempStr, "SETVALIDUNITS:"))
    {
        sscanf(Line, "SETVALIDUNITS: %ld %ld %d", &start, &finish, &value);
        SetUnits(start, finish, value);
        return(TRUE);
    }
    if (!strcmp(TempStr, "SETVALIDAREA:"))
    {
        sscanf(Line, "SETVALIDAREA: %d %d %d %d %d", &x,&y,&tx,&ty, &value);
        SetArea(x,y,tx,ty,value);
        return(TRUE);
    }
    if (!strcmp(TempStr, "SETVALIDANTIAREA:"))
    {
        sscanf(Line, "SETVALIDANTIAREA: %d %d %d %d %d", &x,&y,&tx,&ty, &value);
        SetAntiArea(x,y,tx,ty,value);
        return(TRUE);
    }
    if (!strcmp(TempStr, "SETVALIDDESTYPE:"))
```

```
    {
        sscanf(Line, "SETVALIDDESTYPE: %ld %d", &destype, &value);
        SetDestype(destype, value);
        return(TRUE);
    }
    if (!strcmp(TempStr, "STRIPBADPROXIMITY:"))
    {
        sscanf(Line, "STRIPBADPROXIMITY: %d %d %d", &height, &radius, &max);
        Radius = radius;
        MaxAllowed = max;
        StripBadProximityValues(height);
        return(TRUE);
    }
    if (!strcmp(TempStr, "SETPROXIMITY:"))
    {
        sscanf(Line, "SETPROXIMITY: %d %d", &radius, &max);
        Radius = radius;
        MaxAllowed = max;
        return(TRUE);
    }
    if (!strcmp(TempStr, "FIXBAD:"))
    {
        sscanf(Line, "FIXBAD: %ld", &searchlimit);
        DoubleReplacement(searchlimit);
        return(TRUE);
    }
    if (!strcmp(TempStr, "WEIGHT:"))
    {
        sscanf(Line, "WEIGHT: %d %lf", &radius, &dval);
        ScanRadius = radius;
        LeakageHalfLife = dval;
        weightflag = TRUE; // use weights
        return(TRUE);
    }
    if (!strcmp(TempStr, "NOWEIGHT:"))
    {
        weightflag = FALSE;
        return(TRUE);
    }
    return(FALSE);
}

ChipArrayClass::ReadInstructionFile(char *FileName)
{
        // read the file and be happy
    FILE *fp;
    char dataline[MXLINE];

    fp = fopen(FileName, "rt");
    if (fp==NULL)
        return(FALSE);
    while (fgets(dataline, MXLINE, fp))
    {
                InterpretInstructionLine(dataline);
    }
    fclose(fp);
}


TestTwo()
{
        ChipArrayClass Test;
    Test.ReadInstructionFile("test.opt");
}

Live(char *FileName)
{
        ChipArrayClass Test;
    Test.ReadInstructionFile(FileName);
}
```

```
main(int argc, char **argv)
{
    if (argc==2)
    {
       Live(argv[1]);
    }
    else
    {
       printf("File name required!");
    }
};
```